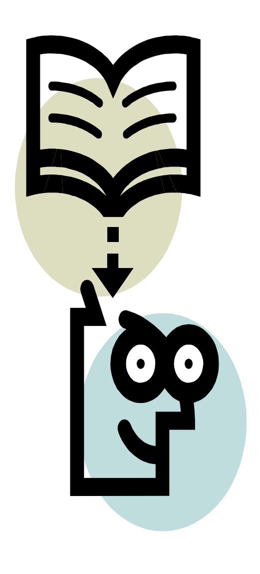
تأليف

الدكتور نضال خضير العبادي

أساسيات البرمجة بلغة البرمجة باسكال



الأهداء

صدقة جاريه ..

الى روح أبي ..

في عليين...

المقدمه

بسم الله الرحمن الرحيم والصلاة والسلام على سيدنا محمد أشرف الخلق وخاتم النبين وعلى أل بيته وصحبه الطيبين الطاهرين.

أما بعد , فها أنذا أضع بين يدي القاريء العربي نتاج جهد بضعة أشهر من العمل الجاد والدؤوب ليكون هذا الكتاب شبه كامل (لأن الكمال لله وحده) لمباديء وأساسيات البرمجه بلغة البرمجه باسكال .

لقد طورت لغة البرمجه باسكال من قبل (Niklaus Wirth) في (Eidgenossische Technische Hochschule in Zurich) وهي مشتقه من لغة البرمجه (Algol 60) ولكن بأمكانيات أفضل مع سهوله بالأستخدام .. وتستخدم لغة البرمجه باسكال الأن بشكل واسع كلغة برمجه مفيده ممكن تنفيذها بكفاءه وهي أداة تعليميه ممتازه .
من المهم أن أذكر بقلة المصادر العربيه في المجالات العلميه بشكل عام والتي تشكل عائقا أمام المعرفه في الوطن العربي , لذا كان هذا الكتاب أسهامه متواضعه عسى أن يكون مفيدا للراغبين بتعلم البرمجه وخصوصا البرمجه بلغة البرمجه باسكال , وتوخيت الأجتهاد في ترجمة المصطلحات الأجنبيه الى اللغه العربيه نظرا لعدم وجود ترجمه عربيه معتمده لمصطلحات البرمجه , وبما يتناسب وسهولة فهمها من قبل القاريء وقربها من المفهوم الحقيقي لها أخذا بنظر الأعتبار ذكر المصطلح الأجنبي مع ما يقابله من ترجمه عند أول ورود له في الكتاب .. وبالتأكيد فأن هذا العمل لايعد مرجعا باللغه العربيه لمصطلحات البرمجه لأن هناك من يكرس جهده ووقته للعمل في هذا المجال .

يبدأ الكتاب بشرح المفاهيم الأساسيه العامه للبرمجه والتي تستخدم مع لغة البرمجه باسكال , وهي بشكل عام مشتركه بين الكثير من لغات البرمجه , أما الفصل الثاني فيتضمن شرحا وافيا لأوامر الأدخال والأخراج وكيفية أستخدامها بشكل كفوء , بينما كرس الفصل الثالث لعبارات القرار والتكرار والتي تعد أحدى الركائز المهمه في البرمجه , وتناول الفصل الرابع المصفوفات بنوعيها الأحاديه والثنائيه , بينما تطرق الفصل الخامس الى الأجراءات والدوال وكيفية أستخدامها , الفصل السادس يشرح بتوسع نسبي الدوال المستخدمه مع السلاسل الحرفيه , وأخيرا تم الختام مع الأنواع والمجموعات .

لقد حاولت جاهدا أن أبسط المواضيع التي تطرقت لها بشكل كبير وأسهبت بعض الشيء في شرحها لأسهل على القاريء التواصل مع مواضيع الكتاب وهي جميعا نتاج خبرتي في تدريس هذه الماده لعدد من السنوات , وأرى أن هذا الكتاب ممكن أن يكون مرجعا جيدا للمبتدئين في البرمجه وكذلك لأصحاب الخبره , وهو ممكن أن يوفر الأساس الجيد لمن يرغب تعلم لغات برمجه أخرى .

وفي كل الفصول تم الأستعانه بأمثله توضيحيه وكانت الفقره الأخيره لكل فصل تحتوي على عدد لابأس به من الأمثله المحلوله والتي تم أنتقائها لتوضيح حالات مختلفه من تقنيات البرمجه أضافه الى التنوع بطرق الحل لهذه الأمثله والأمثله الأخرى ضمن الفقرات الأخرى مما

يساعد القاريء على الألمام بطرق حل مختلفه , وبالرغم من قصر هذه البرامج وبساطتها لكنها كامله وليست أجزاء , وأود أن أاكد أن الكثير من حلول الأمثله لا تمثل الحل النموذجي وذلك لأني ركزت على توضيح تقنيات وطرق حلول مختلفه لزيادة الفائده , وبأمكان القاريء أن يفهم هذه البرامج بشكل جيد من خلال قرائتها , ولكن ننصح بأن تطبق هذه البرامج وتنفذ على الحاسب لزيادة أدراكها وفهمها .

أرجو من الله أن يتقبل مني هذا العمل خالصا لوجهه .

وأني مستعد لسماع أرائكم ومقترحاتكم والأجابه عن أسئلتكم من خلال البريد الألكتروني أدناه :

Comp_dep_educ@yahoo.com

د. نضال العبادي

بغداد / 2006

المحتويات

الفصل الأول – مدخل الى البرمجه بلغة البرمجه باسكال

1.1 المقدمه 1
1.2 بعض الصفات العامه للبرنامج 1
1.3 المعرفات 2
1.4 الثوابت 3
1.5 البيانات 4
1.5.1 الأعداد الصحيحه 4
1.5.2 الأعداد الحقيقيه 5
1.5.3 الرموز 7
1.5.4 السلاسل الحرفيه 9
1.6 التعابير المنطقيه 9
1.6.1 العمليات المنطقيه 9
1.7 توليد الأرقام العشوائي 13

الفصل الثاني – أوامر الأدخال والأخراج

2.1 المقدمه 15
2.2 هيكلية البرنامج 15
2.3 المخرجات والمدخلات 16
2.4 متغيرات السلاسل الحرفيه 24
2.5 أنواع الأخطاء التي تحدث في البرنامج 26
2.6 أمثله محلوله 30

الفصل الثالث – أيعازات القرار والتكرار

3.1 المقدمه 32
3.2 عبارة أذا 32
3.3 أذا المركبه 35
3.4 عبارة التكرار Repeat – Until 36
3.5 عبارة التكرار While – Do 37
3.6 عبارة التكرار For 40
3.7 أستخدام For المتداخله 41
3.8 عبارة أختيار الحاله Case 43
3.9 جملة IN 47
3.10 أمثله محلوله 48

الفصل الرابع – المصفوفات

4.1 المقدمه 53
4.2 المصفوفات 53
4.2.1 المصفوفه الأحاديه 53
4.2.2 المصفوفه الثنائيه 58
4.3 أمثله محلوله 61

الفصل الخامس – الأجراءات والدوال

5.1 المقدمه 66
5.2 الأجراءات 66
5.3 الدوال 75
5.4 أمثله محلوله 77

الفصل السادس – السلاسل الحرفيه

6.1 المقدمه 80
6.2 ماهي السلاسل الحرفيه 80
6.3 العمليات التي تجرى على السلاسل الحرفيه 81
6.3.1 تحديد الموقع 81
6.3.2 الأستنساخ 82
6.3.3 الحذف 83
6.3.4 الحشر 84
6.3.5 دمج سلسلتان حرفيتان 85
6.3.6 حساب طول السلسله الحرفيه 86
6.3.7 تحويل الحروف الصغيره الى حروف كبيره 87
6.3.8 تحويل القيم الرقميه الى سلسله حرفيه 88
6.3.9 تحويل السلاسل الحرفيه الى أرقام 88
6.4 أمثله محلوله 90

الفصل السابع – متغيرات الأنواع

7.1 المقدمه 93
7.2 الأنواع 93
7.2.1 الأنواع العدديه 94
7.2.2 المديات الجزئيه 97
7.2.3 المجموعات 98
7.3 أمثله محلوله 102

الفصل الأول

مدخل الى البرمجه بلغة البرمجه باسكال

1.1 المقدمه

البرنامج هو سلسله متتاليه من الأيعازات , يمكننا تشبيهها بوصفة أعداد وجبه غذائيه , النوته الموسيقيه , أو نموذج حياكه . وتتميز عنها برامج الحاسوب بشكل عام بأنها أطول أمتدادا وكتابتها تستدعي دقه وعنايه فائقتين . وقبل الشروع والخوض في موضوع البرمجه لابد من تعريف بعض المصطلحات التي تأتي لاحقا.

1.2 بعض الصفات العامه للبرنامج

- يحتاج البرنامج بصوره عامه الى من يكتبه وهو المبرمج (Programmer) , والى المعالج (Processor) لتفسير وتنفيذ (Execution OR Running) الأيعازات أو الأوامر (Instructions OR Commands) , وتسمى عملية تنفيذ كامل البرنامج المعالجه (Process)

- أن تنفيذ البرنامج يتم بصوره متتاليه (أي أيعاز (instruction) بعد الأخر حسب تسلسلها), مالم يتم الأخبار خارجيا عن غير ذلك. هذا يعني أن نبدأ بأول أيعاز وينفذ ثم الثاني والثالث وهكذا لحين الوصول الى الأيعاز الأخير. هذا النموذج ممكن أن يغير بطريقه محدده مسبقا بشكل جيد من قبل المبرمج كما يمكن أن يتم تكرار جزء من البرنامج وحسب تحديدات المبرمج (مثلماٍ يتم تكرار مقطع من نوته موسيقيه) .

- أي برنامج يجب أن يكون له تأثير.. مثلا في القطعه الموسيقيه يكون هذا التأثير عباره عن صوت , أما في برامج الحاسوب هذا التأثير يكون على شكل مخرجات , أما مطبوعه أو معروضه على الشاشه .

- كل برنامج يعمل على أشياء محدده للوصول الى التأثير المطلوب (مثلا في وصفة أعداد الطعام فان هذه الاشياء ممكن أن تكون اللحوم , الخضار, وغيرها) , أما في البرامج فأن هذه الاشياء تكون بيانات .

- في العديد من البرامج يجب أن يتم الأعلان المسبق عن المتغيرات أو البيانات التي سيتم أستخدامها وماهية أنواعها (هذا مشابه لعملية أعداد وجبة طعام حيث يجب أن تحتوي الوصفه ابتداءا ماهية المواد التي ستستخدم وكمياتها) .

- في بعض الأيعازات ربما تكون هناك حاجه أن يترك أتخاذ قرار تنفيذها الى المعالج وفقا لشروط معينه.. فمثلا (عند أعداد وجبة طعام يكتب في الوصفه ما يلي " عند توفر الطماطه الطازجه تستخدم بعد نزع القشر في خلاف ذلك يستخدم معجون الطماطم") .
- ربما تكون هناك حاجه لتنفيذ أيعاز أو مجموعه من الأيعازات لأكثر من مره . عليه طالما هناك أيعاز يراد تكراره فأن عدد مرات التكرار يجب ان تحدد . ممكن أنجاز ذلك أما بتحديد

عدد مرات التكرار (مثلا يوضع الطعام على النار لمدة 30 دقيقه) أو بفحص حاله تكون من ضمن العمليه (مثلا وضع الطعام على النار لحين أن ينضج) .

1.3 المعرفات IDENTIFIERS

كل البرامج تحتوي على نوعين من الرموز :

النوع الاول .. وهي ***الرموز التي تعود الى اللغه*** .. ففي لغة البرمجه باسكال تستخدم هذه الرموز بطريقتين أما أن تكون على شكل رمز واحد أو أثنين مثل (- , + , = : , ( ) , ; , :) أو على شكل كلمات تسمى الكلمات المحجوزه مثل :

(begin , if , else , repeat , while, until , then , end)

النوع الثاني .. هو **المعرفات** وهي عباره عن رموز تستخدم في البرامج فأما أن تكون معرفات قياسيه مثل

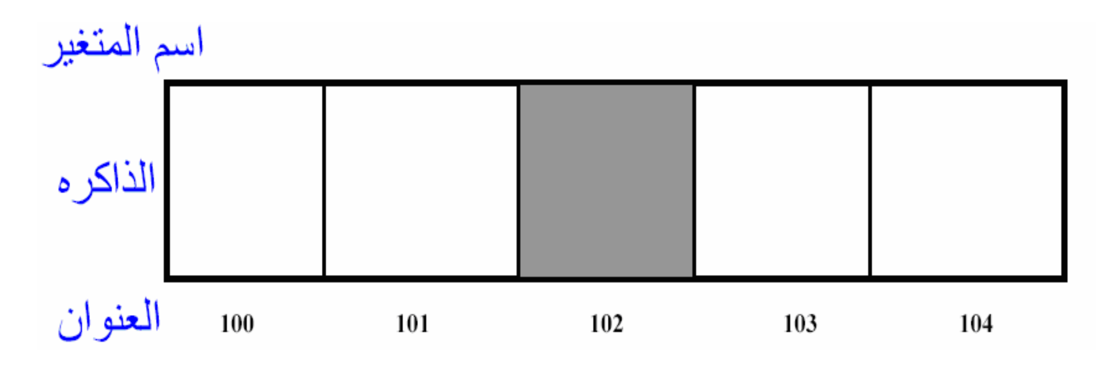
(integer , real , write, sqrt …etc)

أو أن تكون معرفات يتم أختيارها من قبل المبرمج . هذه المعرفات الأخيره نسميها أيضا المتغيرات (variables) و**المتغير** هو رمز أو أكثر يستخدم في البرنامج ليشير الى محتوى موقع في الذاكره .

المتغير ://

في أغلب لغات البرمجه فأن المتغير هو مكان لتخزين المعلومات , المتغير هو مكان أو موقع في ذاكرة الجهاز حيث يمكن تخزين قيمه بداخله ثم أمكانيه أستعادة هذه القيمه فيما بعد .
والمتغير هو أسم يمثل برقم أو سلسله حرفيه (وممكن حرف واحد أو تعبير منطقي) .

من الممكن تصور ذاكرة الجهاز على أنها مجموعه من المواقع التي تخزن فيها المعلومات , هذه المواقع مرقمه بشكل متسلسل تبدأ من الصفر وتنتهي بحجم الذاكره , تعرف هذه الأرقام بعناوين الذاكره سيمثل أسم المتغير بطاقة عنونه ملصقه على أحد المواقع بحيث تستطيع الوصول اليه سريعا دون الحاجه الى معرفة العناوين الحقيقية في الذاكرة (لذا فان المتغير سيشير الى احد هذه العناوين وعند وضع قيمه في المتغير فان المترجم (compiler) سيذهب الى العنوان الذي يشير له المتغير ويضع فيه القيمه وكذلك عندما نريد أن نعرف قيمة المتغير فأن المترجم يذهب الى العنوان الذي يشير له المتغير ويقرأ القيمه التي فيه) . يعرض الشكل التالي هذه الفكره والتي تبين بعض المواقع في الذاكرة والتي ممكن ان يشير اليها المتغير .



شكل رقم (1.1) : بعض مواقع الذاكره

تتكون المتغيرات من حرف واحد , مجموعة حروف , أو حروف وأرقام على أن يكون **أول رمز حرف** مثل
(x , b , ad, jasim , endofpoint , hind6 , x345) هذه جميعا متغيرات مقبوله.

أما المتغيرات التاليه فهي غير مقبوله (first name , next.word , 15may) والسبب هو أن المتغير الأول يحتوي على فراغ والثاني يحتوي على نقطه أما الأخير فهو يبدأ برقم وهذه جميعها غير مقبوله في البرنامج .
أن أول ظهور للمتغير يكون في قسم الأعلان عن المتغيرات حيث يتم الأعلان عن المتغير ويحدد نوعه (أي هل هو رقم صحيح (integer) مثلا , كسر (real) , نص كتابي (string) , حرف (character)...الخ) . أن أختيار المتغير من قبل المبرمج تعتبر مسأله مهمه ويفضل أن يعكس المتغير المعنى الذي يستخدم لأجله المتغير فمثلا يفضل أستخدام المتغير (sum) مع الجمع وأذا ما أستخدم متغير أخر فان ذلك سوف لا يؤدي الى أي اشكال , وكذلك يفضل أن لا يكون المتغير طويل فمثلا يفضل أستخدام متغير متكون من حرف واحد عندما نستخدمه في برنامج قصير ولا يتكرر كثيرا , أما أستخدام متغير من حرف واحد ويستخدم بشكل متكرر وبأجزاء متكرره في برنامج طويل فأنه يعتبر أختيار سيء بالرغم من أنه لا يعيق عمل البرنامج.

1.4 الثوابت CONSTANTS

في بعض البرامج نحتاج الى أستخدام قيم ربما تكون معروفه مسبقا قبل تنفيذ البرنامج ولا يمكن أن تتغير داخل البرنامج مثل النسبه الثابته (Л) والتي تكون قيمتها (3.1415926585) هذه القيم الثابته سواء كانت ذات قيمه معروفه مسبقا أو أي قيمه ممكن أن تسند الى متغير , جميعها ممكن أن تعرف بحقل خاص يدعى حقل الأعلان عن الثوابت وهذا الحقل يسبق حقل الأعلان عن المتغيرات وكما يلي :

```
Const
   Pi = 3.1413926535 ;
  Error = ' Run_Time Error ' ;
```

أسباب أستخدام الثوابت :

- أذا كان هناك عدد يستخدم بشكل متكرر داخل البرنامج فأن المبرنج يفضل أن يصفه بأسم ضمن حقل الأعلان عن الثوابت , وبعدها بالأمكان أستخدام الأسم الموصوف في القسم التنفيذي .
- من الممكن أستخدام حقل الثوابت لتسمية متغيرات من نوع السلاسل الحرفيه والتي تستخدم بشكل متكرر في مخرجات البرنامج .

مثال : نفرض أننا نحتاج الى طباعة أسم جامعه مثلا بشكل متكرر في البرنامج , ممكن أن نقوم بمايلي :

```
Const
    University = 'Al _ Mustnsirah university ' ;
    Underline = '---------------------------------------------' ;
```

الأن من الممكن أستخدام الأسماء المعرفه كثوابت في البرنامج وكما يلي:

```
Writeln ( university ) ;
Writeln ( underline ) ;
```

1.5 البيانات DATA

كل عنصر من البيانات في البرنامج أما أن تكون قيمته ثابته أو متغيره (أن قيمة المتغير ربما تتغير خلال تنفيذ البرنامج). كل متغير(والذي هو بيانات) في البرنامج يجب أن يكون له نوع وبموجب هذا النوع سيتم تحديد المسا حه الخزنيه اللازمه لقيمة هذا المتغير , وكذلك تحدد العمليات التي ممكن أجراؤها على هذا المتغير. والأنواع القياسيه التي تستخدم في لغة البرمجه باسكال هي :

1.5.1 الاعداد الصحيحه INTEGERS

الأعداد الصحيحه هي كل الأعداد الموجبه والسالبه التي لا تحتوي على كسر. فالصفر عدد صحيح و 567 هو عدد صحيح و 23- أيضا عدد صحيح . أما (123.345 و 1.45-) فهي ليست أعداد صحيحه. أن اعلى قيمه وأوطأ قيمه لعدد صحيح ممكن تمثيله في الحاسوب تختلف من حاسوب لأخر , ويمكن معرفة هذه القيم في أي حاسوب بأسخدام الأيعازات التاليه :

لمعرفة أعلى قيمه نستخدم	**maxint**
لمعرفة أوطأ قيمه نستخدم	**- maxint**

أن أي محاوله لأستخدام قيم خارج نطاق الحدود العليا والدنيا سيؤدي الى حدوث خطأ. وبشكل عام فأن المتغيرات من نوع الأعداد الصحيحه تستخدم أضافه الى العمليات الرياضيه في العدادات والفهارس .

العلاقات الرياضيه التي تستخدم مع الأعداد الصحيحه هي (+ , - , * , Div , Mod) وهي على التوالي (الجمع , الطرح , الضرب , القسمه , وحساب باقي القسمه) .

يجب أن نلاحظ هنا أن العلامه (/) تستخدم للقسمه لكن مع الأعداد الحقيقيه أي التي تحتوي كسور

أمثله ://

2 Div 3 = 0
2 / 3 = 0.66666667
5 Div 1 = 5
5 / 1 = 5.0
5 Div 2 = 2
2+3*4 = 14
(2+3) * 4 = 20 هنا ينفذ داخل القوس أولا
5 mod 2 = 1
7 mod 4 = 3

ويصرح عن الأعداد الصحيحه بلغة البرمجه باسكال في حقل الأعلان عن المتغيرات (سنوضحها لاحقا) بالداله (integer) .

1.5.2 الأعداد الحقيقيه REAL NUMBERS

وهي الأعداد التي تحتوي على كسور مثل 0.03 , 12.5 , -356.67890 , 10.0

أما العمليات الرياضيه التي ممكن أجراؤها عليها فهي (+ , - , * , /) وهي (الجمع , الطرح , الضرب , القسمه). ويصرح عن الأعداد الحقيقيه في بلغة البرمجه باسكال في حقل الأعلان عن المتغيرات بالداله (Real) .

ملاحظه ://

تمثل الأرقام بطريقتين فأما أرقام صحيحه بدون كسر أو أرقام كسريه . القواعد التاليه تطبق عند كتابة أرقام

1. **الفارزه لا يمكن أن تظهر في أي مكان في الرقم .**
2. **ممكن أن تسبق الأرقام أحدى العلامتين (- , +) للدلاله على كونه موجب أو سالب (يعتبر الرقم موجبا أذا لم تظهر أي من العلامتين أمامه) .**
3. **يمكن تمثيل الأرقام بطريقة العلامه العلميه (وذلك بأستبدال الرقم (10) بالحرف (E)) . مثلا الرقم (2.7×10^{-6}) تكتب حسب العلامه العلميه كما يلي (2.7E-6) .**

ملاحظه ://:

أدناه بعض القواعد الهامه التي يجب أن تراعى عند كتابة العلاقات الرياضيه :

1. **أن وضع أشارة السالب قبل المتغيرات هي مكافئه لضرب المتغير بالقيمه (1-) . مثلا المتغيرات (x+y) – من الممكن أن تكتب (x+y) * 1- .**
2. **يجب أن تكتب العلاقات الرياضيه وفقا للطريقه التي تحددها لغة البرمجه باسكال بحيث تذكر كل العلامات الرياضيه دون أختصار . مثال : العلاقه الرياضيه الأتيه غير مقبوله ((2(x1 + 3x2) هذه العلاقه لكي تكون مقبوله في لغة البرمجه باسكال يجب أن تكتب بالشكل التالي : ((2 * (x1 + 3 * x2) العلاقه الأولى هي التي تعودنا على أستخدامها في الرياضيات .**
3. **الرقم السالب ممكن أن يرفع الى أي أس بشرط أن يكون الأس عدد صحيح (لأن الرقم المرفوع الى قيمه معينه سيضرب بنفسه عدد من المرات بقدر الأس أذا كان عدد صحيح ولا يهم فيما أذا كان الأساس سالب أو موجب) .**
4. **لايجوز رفع القيمه السالبه الى أس كسري (وذلك لأن حساب ناتج الرقم المرفوع الى أس كسري يتم بحساب اللوغاريثم للأساس , ويضرب هذا اللوغاريثم بالأس , وعندها يحسب معكوس اللوغاريثم , وحيث أن اللوغاريثم للرقم السالب غير معرف لذا لايمكن أيجاد النتيجه) .**
5. **العمليات الرياضيه لايمكن أجراؤها على السلاسل الحرفيه . مثال ('xyz' + 34) هذا غير مقبول وذلك لأن (xyz) هو سلسله حرفيه وليس رقم أو متغير رقمي (لاحظ أنه محصور بين علامتي أقتباس (quotation mark) للدلاله على أنه سلسله حرفيه) .**

ملاحظه ://:

ممكن أستخدام قيم الأعداد الصحيحه في التعابير الرياضيه التي تستخدم الأعداد الحقيقيه (ولا يمكن العكس) , فأذا كانت أحدى القيم لأي من العمليات (* , – , +) قيمه حقيقيه فأن القيمه الأخرى تحول الى حقيقيه أليا قبل تطبيق العمليه .
أما عند أستخدام عملية القسمه (/) فيجب أن يكون كلا القيمتين حقيقيتين .

1.5.3 الرموز Characters

وهي كافة الرموز التي تستخدم في الحاسوب والتي غالبا ما نجدها على لوحة المفاتيح والتي تشمل الحروف الأبجديه سواء كانت حروف كبيره (A . . Z) أو حروف صغيره (a..z) , الأرقام (0..9) , الرموز الاخرى التي نراها على لوحة المفاتيح مثل (+ , . , / , ? , # , ! , & , % ..etc) وتستخدم بشكل مفرد . ويصرح عن الرموز بلغة البرمجه باسكال في حقل الأعلان عن المتغيرات بالداله (char) . ولا توجد هناك مجموعه خاصه من الرموز للغة البرمجه باسكال لأن لغة البرمجه باسكال تستخدم مجموعة الحروف للحاسوب الذي تعمل عليه .

أن أكثر مجاميع الحروف أستخداما هما أثنان :

2 ASCII

(American Standard Code for Information International)

3 EBCDIC

(Extended Binary Coded Decimal Information Code)

وكل منهم له صفاته الخاصه به .

ملاحظه ://

كل ما يكتب بين علامتي أقتباس (' ') هو واحد من أثنين :

1. **أذا كان أكثر من رمز واحد فيعتبر سلسله حرفيه .**
2. **أذا كان رمز واحد فيعتبر حرف وممكن في بعض الحالات يعتبر سلسله حرفيه أذا كان معرف كذلك .**

مجموعة الحروف لها الخواص والصفات التاليه :

1. كل حرف له عدد ترتيبي (Ordinal Value) مختلف , حيث أن هذه الحروف مرتبه وفقا لأحد النظامين أعلاه .
وتستخدم الداله (Ord) لتعيد قيمة العدد الترتيبي , فمثلا أذا كان كل من (ch1 , ch2) هما من نوع حروف (char) وكان

$$ch1 \neq ch2$$

فأن

$$ord (ch1) \neq ord (ch2)$$

ملاحظه ://

العوامل الوحيده التي تستخدم مع المتغيرات الحرفيه هي:

(<, <= , = , >= , >)

Ord ('1') , ord ('2') , ord ('3') , …….., ord ('9')

ملاحظه ://

في غالبية المجاميع الحرفيه فأن ord (' 0 ') $\neq$ 0 لذا فأن الداله (ord) لا تحول الأرقام الى القيمه المقابله .

3. لكي نحول الرقم بالمتغير الحرفي (ch) الى القيمه الرقميه المقابله (num) فيجب أستخدام الصيغه التاليه :

Num : = ord (ch) – ord (' 0 ') ;

4. الأعداد الترتيبيه للأحرف الكبيره (A , B , ….. , Z) يجب أن تكون مرتبه وليس بالضروره أن تكون متعاقبه .

5. الأعداد الترتيبيه للأحرف الصغيره (a , b , ….. , z) (أذا وجدت) يجب أن تكون مرتبه وليس بالضروره أن تكون متعاقبه .

أن الفقرتين (4 و 5) تؤكدان على أن تكون الحروف مرتبه هجائيا , ولكن ليس بالضروره أن تكون لها أعداد ترتيبيه متعاقبه , فمثلا في نظام (EBCDIC) :

Ord (' I ') = 201
Ord (' J ') = 209

6. الداله (chr) تعمل عكس الداله (ord) فهي تأخذ معامل من نوع عدد صحيح وتعطي القيمه الحرفيه المقابله له , ومدى عمل هذه الداله هو مدى عمل الداله (ord).

num : = ord (ch) ;
ch : = chr (num) ;

7. عمليا أذا ما أخذنا رقم (num) من نوع الأعداد الصحيحه ($0 \leq$ num $\leq$ 9) فأن الحرف المقابل له هو وفقا للصيغه التاليه :

ch : = chr (num + ord (' 0 ')) ;

مثلا :

chr (3 + ord (' 0 ')) = ' 3 '

1.5.4 السلاسل الحرفيه STRING

وهي عباره عن متواليه من الرموز المبينه في الفقره السابقه , كذلك يمكن أن تحتوي السلاسل الحرفيه على فراغ ويعتبر رمز ولكن لا يمكن أن تحتوي السلاسل الحرفيه على علامات الأقتباس . أن السلاسل الحرفيه تستخدم لتعريف المعلومات غير الرقميه مثل الأسماء , العناوين وغيرها, وأن عدد الحروف التي تستوعبها السلسله الحرفيه بلغة البرمجه باسكال هي (255 .. 0) . ويصرح عن السلسله الحرفيه في لغة البرمجه باسكال في حقل الأعلان عن المتغيرات بالداله (string) . **مثال**

'Xyz'
'Ali Abbas'
'Apollo-17'
'Do you wish to try again'

الجمله الأخيره تعتبر سلسله حرفيه , كذلك فأن الأرقام عندما تعرف مع السلاسل الحرفيه تعامل كرموز وليس أرقام .

1.6 التعابير المنطقيه THE BOOLEAN EXPRESSIONS

وهي التعابير التي تمثل نتيجتها بحاله واحده من أثنتين وهما (صح أو خطأ) (**true OR false**) , وهناك ثلاث عوامل منطقيه وهي (**And , Or , Not**) .
والتعبير المنطقي يعيد القيمه (1) عندما يكون التعبير (TRUE) والقيمه (0) عندما يكون التعبير (FALSE) . وهي تستخدم لوصف أي تعبير فيما أذا كان صح أو خطأ . أن أنواع المتغيرات التي تستخدم لهذا الغرض يصرح عنها في حقل المتغيرات بالداله (Boolean).

1.6.1 العمليات المنطقيه LOGIC OPERATORS

هناك ثلاث أنواع من العمليات المنطقيه وهي (**AND , OR , NOT**) كل منها يتعامل مع التعابير الشرطيه (أي التي تحتوي شرط). كل واحد من هذه التعابير له تأثير مختلف على التعابير الشرطيه . أدناه أمثله تبين كيفية أستخدام هذه التعابير والتي من الممكن أن تستخدم بين تعبيرين أو أكثر من التعابير الشرطيه.

- **AND**

```
If  (Str1 = 'a')  AND  (Str2 = 'b')  then

 Writeln ( 'Yes, you got it right.' ) ;
```

جدول (1.1) : جدول الصدق للعامل (و) (And)

Expression 1	Expression 2	AND (result)
true	true	*true*
false	true	false
true	false	false
false	false	false

- **OR**

```
If  (Str1 = 'a')  OR  (Str2 = 'b') then

writeln('Yes, you got it right.');
```

جدول (1.2) : جدول الصدق للعامل (أو) (Or)

Expression 1	Expression 2	OR (result)
true	true	*true*
false	true	*true*
true	false	*true*
false	false	false

- **NOT**

جدول (1.3) : جدول الصدق للعامل (لا) (Not)

Input	Output
true	false
false	true

ملاحظه ://

العامل (NOT) يختلف عن العاملين السابقين حيث أنـه يتقبل مدخل واحد ودائما يعكس حالة العبـاره التي يدخل عليها فـأذا كانت صحيحه فيجعلها خاطئـه وأن كانت خاطئه يجعلها صحيحه.

جدول (1.4) : أهم أنواع المتغيرات المستخدمه بلغة البرمجه باسكال

النوع	المدى	الحجم (بايت)	الملاحظات
Byte	0 .. 255	1	أعداد صحيحه موجبه فقط
Word	0 .. 65535	2	أعداد صحيحه موجبه فقط
Shortint	-128 .. 127	1	أعداد صحيحه
Longint	-2146473648 … 2146473647	4	أعداد صحيحه
Real	$2.9x10^{-39}$ .. $1.7x10^{38}$	6	أعداد حقيقيه
Sigle	$1.5x10^{-45}$ .. $3.4x10^{38}$	4	أعداد حقيقيه
Double	$5.0x10^{-324}$ .. $1.7x10^{308}$	8	أعداد حقيقيه
Extended	$3.4x10^{-4932}$ .. $1.1x10^{4932}$	10	أعداد حقيقيه
Char	حرف واحد	1	غير رقميه
String	255 حرف	255	غير رقميه
Boolean	True / False	1	غير رقميه

ملاحظه ://

أن أسناد قيمه لمتغير من نوع معين خارج المدى المحدد له سيؤدي الى حدوث خطأ , هذا الخطأ أما أن يوقف التنفيذ أو يؤدي الى ظهور نتائج غير متوقعه .

جدول (1.5) : بعض الدوال المهمه المستخدمه مع لغة البرمجه باسكال

الداله	الوظيفه	مثال
abs	لأيجاد القيمه المطلقه لعدد سالب (يحول العدد السالب الى موجب)	Abs (-5) = 5
sqr	أيجاد مربع عدد	Sqr (5) = 25
sqrt	أيجاد الجذر التربيعي لعدد موجب	Sqrt (25) = 5.0000000000E+00
sin	ايجاد جيب زاويه	Sin (30 * Pi / 180) = 5.0000000000E-01
cos	أيجاد جيب تمام زاويه	Cos (60 * Pi / 180) = 5.0000000000E-01
trunk	تحول الأعداد الكسريه الى أعداد صحيحه وذلك بحذف الكسر	Trunk (5. 2431) = 5 Trunk (-5.6) = -5
round	يقرب العدد الكسري الى أقرب عدد صحيح	Round (5.2431) = 5 Round (5.765) = 6 Round (-5.8) = -6
Int	يحول العدد الكسري الى عدد صحيح	Int (2.31) = 2.0000000000E+00
inc	يعمل على زيادة المتغير بقدار واحد أو حسب ما محدد	نفرض أن قيمة (x) هي x = 10 عليه فأن (inc (x)) ستؤدي الى x = 11 أما (inc (x, 4)) ستؤدي الى x = 14
dec	تقليص قيمة المتغير بمقدار واحد أو حسب ما محدد	نفرض أن قيمة (x) هي x = 15 عليه فأن (dec (x)) ستؤدي الى x = 14 أما (dec (x , 5)) ستؤدي الى x = 10
pred	تعطي القيمه السابقه للمتغير	أذا كانت (x = 10) فأن (pred (x) = 9)
succ	تعطي القيمه اللاحقه للمتغير	أذا كانت (x = 10) فأن (succ (x) = 11)
odd	يفحص المتغير فيما أذا كان عدد فردي أم لا ونتيجته تكون صح أو خطأ	أذا كانت (x = 9) فأن (odd (x)) ستكون (true)
ln	تعطي قيمة اللوغاريتم الطبيعي (أي للأساس (e))	Ln (8) = 2.0794415417E+0

ملاحظه :

هناك نوعان من الدوال .. الدوال القياسيه وهي الدوال المعرفه ضمن لغة البرمجه مثل الدوال في الجدول (1.5) , ودوال تعرف من قبل المستخدم والتي سنشرحها في الفصل الخامس .

ملاحظه ://:

تقاس الزوايا في الدوال التي تستخدم الزوايا (في لغة البرمجه باسكال) بما يسمى بالنصف قطريه (Radians) وليس بالدرجات كما هو الحال في الرياضيات , ولتحويل أي زاويه (angle) من القياس بالدرجات الى النصف قطريه نتبع العلاقه التاليه :

Angle (in radians) : = angle (in degree) * Pi / 180

حيث أن (Pi) هي النسبه الثابته وهي من القيم المخزونه في لغة البرمجه باسكال وسيعوض عنها أليا بقيمتها البالغه (3.14159265358979323850) .

ملاحظه ://:

Ord (true) = 1
Ord (false) = 0
Pred (true) = false
Succ (false) = true

1.7 <u>توليد الأرقام العشوائي RANDOM NUMBERS GENERATION</u>

تحتاج بعض التطبيقات الى أستخدام أرقام عشوائيه وهذا ممكن في لغة البرمجه باسكال وذلك من خلال أستخدام الأمر (Random) الذي يعمل على توليد رقم بشكل عشوائي , وهو يعمل وفقا لما يلي :

- يستخدم مع الأمر (Randomize) حيث يجب أن يسبق أستخدام الأمر (Random) كتابة الأمر (Randomize) , وبذلك فأن الأمر (Random) سيولد أرقام عشوائيه تتراوح قيمها بين الصفر والواحد (غير داخل) (أي أرقام كسريه موجبه قيمتها أقل من واحد) **مثال** :

```
Randomize ;
X : = random ;
```

هنا المتغير (x) تكون قيمته (0 <= x < 1) وفي كل مره يتم تنفيذ هذا الأمر سنحصل على قيمه جديده ضمن نفس المدى .

- الطريقه الثانيه هي بأستخدام الأمر (Randomize) أيضا ثم الأمر (Random) على أن يحتوي الأمر (Random) على المدى المطلوب أيجاد الرقم العشوائي ضمنه (أي أنه سيولد أعداد صحيحه موجبه عشوائيا تتراوح قيمها بين الصفر والعدد المحدد بين القوسين بعد (Random والذي يمثل الحد الأعلى) , **مثال** :

```
Randomize ;
X : = random ( 100 ) ;
```

هنا تكون قيمة المتغير (x) (0 <= x < 100) وفي كل مره يعاد تنفيذ هذا الأمر سنحصل على قيمه جديده . أن المدى المحدد يمكن تغيره حسب طبيعة التطبيق المراد تنفيذه .

- الطريقه الثالثه لأستخدام الأمر (Random) هي بدون أستخدام الأمر (Randomize) وبدلا منه نستخدم المتغير (Randseed) قبل الأمر (Random) على أن يتم أسناد قيمه للمتغير (Randseed) , هذه الطريقه من وجهة نظري هي الأفضل لأن الطريقتين السابقتين ستولدان نفس مجموعة القيم عند أيقاف البرنامج وأعادة تنفيذه مما لا يؤدي الى عشوائيه حقيقيه , بينما هذه الطريقه ستولد مجموعة أرقام عشوائيه مختلفه في كل مره يتم فيها أعادة التنفيذ على أن يتم أسناد قيم مختلفه للمتغير (Randseed) عند كل تنفيذ , **مثال**

```
Randseed : = 1200 ;
X : = random ;

OR

Randseed : = 3425 ;
X : = random ( 1000 ) ;
```

في الحاله الأولى فأن المتغير (randseed) أسند له قيمه وهي (1200) ووفقا لها سيولد أرقام عشوائيه كسريه أقل من واحد ولو أعدنا التنفيذ مع أسناد قيمه مختلفه للمتغير (randseed) فأن رقم عشوائي مختلف ستولد (حاول تنفيذ الطريقتين ولاحظ الفرق) .
أما المثال الثاني فأنه سيولد أرقام عشوائيه أكبر من الصفر وأصغر من (1000) .

الفصل الثاني

أوامر الأدخال والأخراج INPUT / OUTPUT INSTRUCTIONS

2.1 المقدمه

جميع اللغات الطبيعيه التي يتعامل بها الأنسان كوسيله للتخاطب والتواصل لها قواعد وضوابط تحدد ألية أستخدامها , ولما كانت لغات البرمجه تصنف على أنها من اللغات العليا (أي اللغات القريبه من لغات البشر) فكان لا بد وأن تكون لها قواعد تحدد ألية أستخدامها لتكون واضحه للمتعامل معها وكذلك للمترجم داخل الحاسوب . عليه فأن هذا الفصل والفصول الاحقه ستوضح هذه القواعد وسنبدأ بمعرفة كيفية تلقيم الحاسوب بالمعلومات وطرق الحصول على النتائج بعد أنجاز عمليات الحساب خلال هذا الفصل .

2.2 هيكليه البرنامج PROGRAM CONSTRUCTION

يتكون البرنامج بلغة البرمجه باسكال من (الرأس والجسم) (head and block) والرأس هو السطر الأول في البرنامج ويبدأ بكلمه برنامج باللغه الانكليزيه ويتبع بأسم البرنامج ثم الفارزه المنقوطه (;) وكما يلي :

```
Program  programname ;
```

لاحظ هنا يجب ان يكون هناك فراغ بين كلمة برنامج وأسم البرنامج , أما أسم البرنامج فيتم أختياره من قبل المبرمج , وأستخدام الفارزه المنقوطه تدل على نهاية العباره وهي تستخدم مع كل العبارات في برامج باسكال للدلاله على نهاية العباره وبداية عباره جديده عدا بعض الحالات التي سيتم الاشاره لها في حينها.

ملاحظه :

يجب أن تظهر الفارزه المنقوطه في :

1. **بعد عبارة رأس البرنامج , الأجراء , أو الداله .**
2. **بعد كل قائمة تعريفات موجوده في حقل الأعلان عن الثوابت , النوع , والمتغيرات .**
3. **بين العبارات الكامله في الجزء التنفيذي من البرنامج .**

ملاحظه :

من الممكن أهمال الفارزه المنقوطه بعد العبارات التي تسبق الأمر (end) .

أما جسم البرنامج فيتكون من قسمين ***الأول*** هو **قسم الأعلان** عن الثوابت والمتغيرات وغيرها والتي سنأتي عليها لاحقا وهو يستخدم حسب الحاجه اليه أي ممكن أن نرى برامج ليس فيها هذا القسم بسبب عدم أحتياج البرنامج للأعلان . ***والقسم الثاني*** فهو يمثل **(الأيعازات أو الأوامر) (instructions or commands)** التي سينفذها البرنامج وهو يبدأ بكلمة أبدأ باللغه الانكليزيه (**begin**) ثم الأيعازات الواجب تنفيذها , وينتهي بكلمه نهايه باللغه الانكليزيه متبوعه بنقطه (**end .**) وعليه فسيكون شكل البرنامج كما يلي :

```
Program  programname ;

    Declaration section    (may or may not contain   Type ,
                            Const , Var , Uses , Label )
Begin
      Statements  ( instructions ) ;
End.
```

ملاحظه ://:

يفضل عند تسمية البرنامج أن يكون الأسم يدل على فعل البرنامج أو وظيفته , ولا يجوز أن يكون الأسم أحدى الكلمات المحجوزه , أو يكون أسم لمتغير مستخدم في البرنامج , ولا يجوز أن تستخدم المسافات (الفراغات) بين أحرف البرنامج . ممكن استخدام الشارحه (_) او الأرقام مع أسم البرنامج .

ملاحظه ://:

يمكن الأستغناء عن رأس البرنامج وسينفذ البرنامج بشكل طبيعي .

2.3 المخرجات والمدخلات INPUT / OUTPUT

في كل برنامج يجب أن تكون هناك مخرجات تبين النتائج التي تم الحصول عليها من البرنامج , هذه النتائج سيتم عرضها على شاشة الحاسوب بأستخدام الأمر (**writeln**( ) OR **writeln**( )) أن الأمر (write ( )) يعني أكتب ماموجود بين القوسين على هذا السطر الذي تؤشر عليه . أما الأمر (writeln ( )) فهي تعني أكتب على سطر جديد العباره التي بين القوسين , وهذه مفيده في عملية تنظيم المخرجات.
في جميع الأحوال سواء أستخدمنا (write( ) OR writeln ( )) فأننا نحتاج ألى وضع ماملطلوب أظهاره على الشاشه بين القوسين . أن ما يوضع بين القوسين سيأخذ حاله من أثنتين :

2.3.1 ان يكون ما بين القوسين محصور بعلامات أقتباس مفرده (single quotation mark) (' ') وبهذه الحاله فان ما موجود بين علامتي الأقتباس سيتم طباعته على الشاشه دون أدنى تغيير وكما هو. **مثال**

```
Program Ch2_Program1;
Begin
  Write('Hello World. Prepare to learn PASCAL!!');
End.
```

ملاحظه ://

ان البرنامج بلغة البرمجه باسكال لا يتأثر أو يتحسس لشكل الحروف سواء كانت مكتوبه بالأحرف الكبيره أو الأحرف الصغيره .

لاحظ مايلي : //

أولا / عدم أستخدام أو وجود قسم الأعلان وذلك لعدم وجود ما نعلن عنه.

ثانيا / أن مخرجات هذا البرنامج هي العباره الموجوده بين القوسين بعد كلمه (write) وستظهر على الشاشه كما يلي :

مخرجات البرنامج :

Hello World. Prepare to learn PASCAL!!

ثالثا / عند تنفيذ هذا البرنامج سوف لا يمكن ملاحظة المخرجات والسبب هو أن الحاسب سريع جدا بحيث يعرض ويخفي شاشه التنفيذ دون أن نلاحظ ذلك , ولغرض رؤية المخرجات فيمكن بعد ان يتم التنفيذ ضغط الزرين **(Alt+ F5)** معا وعندها ستظهر شاشه التنفيذ (**السوداء**) .. ويمكن الخروج من شاشة التنفيذ بضغط الزر (**Enter**) .

ملاحظه ://

ممكن أظهار النتائج بعد أمر الطباعه بطريقه تساعد على تنظيم المخرجات وذلك من خلال تحديد عدد المواقع (كل حرف أو رقم يطبع على موقع واحد) التي ستطبع عليها النتائج حيث ستطبع القيمه المطلوب طباعتها من اليمين الى اليسار , فأذا كان عدد المواقع أكبر من عدد الارقام في العدد فسيترك فراغ من اليسار , والصيغه العامه هي :

```
Write ( data : fieldwidth) ;
```

لاحظ أن الفاصل بين البيانات والرقم الذي يمثل عرض الحقل هو النقطتان المتعامدتان.

مثال

```
I : = 2345 ;
Writeln ( I ) ;
Writeln  ( I : 7 ) ;
```

ستكون نتيجه هذين الأمرين كما يلي , لاحظ الفرق

```
2345
***2345
```

هنا العلامه (*) تمثل فراغ للتوضيح فقط .

أضافه الى أمكانية تحديد عدد المواقع المخصصه للرقم فيمكن تحديد عدد المراتب بعد الفارزه لطباعة الرقم الكسري وكما يلي :

```
Write ( data  : fieldwidth :  precision  ) ;
```

حيث أن (precision) تمثل عدد المواقع المحدده لطباعة الجزء الكسري . مثال

```
Writeln ( I  :  9  :  3  ) ;
```

يجب أن تكون (I) عدد حقيقي .

في جميع الأمثله أعلاه فأن المتغير (I) معرف كعدد صحيح , أما أذا عرف كعدد حقيقي فأن النتائج ستختلف وتكون للمثال الأول كما يلي :

```
2.3450000000E+03
2.3E+03
```

أما بالنسبه للمثال الأخير فتكون النتيجه

```
2345.000
```

وأخيرا ممكن أن يكون (I) من نوع السلاسل الحرفيه وبهذه الحاله فأن حقل عدد المراتب بعد الفارزه سوف لا يكون له حاجه .

2.3.2 أما أذا كان ما موجود بين القوسين بعد كلمة (write OR writeln) ليس محصور بين علامتي أقتباس فعند ذلك سيعامل ما موجود بين القوسين على أنه معرف والمعرفات يجب أن تكون لها قيمه لذا فان الحاسوب سيطبع قيمة المعرف على شاشة التنفيذ .

هنا علينا أن نلاحظ أن أستخدام أي معرف داخل البرنامج يحتاج الى شرطين :

الأول / أن يتم الأعلان عن المعرف في قسم الأعلان عن المعرفات آلذي سبق أن نوهنا عنه فأذا كان ثابت يعلن عنه في قسم الأعلان عن الثوابت (const) أما أذا كان متغير فيعلن عنه في قسم الأعلان عن المتغيرات (var) ويحدد نوعه . وللأعلان عن متغير في قسم المتغيرات نتبع مايلي:

بدءا نكتب كلمة **var**

الأن نكتب أسم المتغير ويتبع بالنقطتين المتعامدتين (colon) (:) ثم نوع المتغير وتنتهي بفارزه منقوطه

x : integer ;

هذا المتغير هو من نوع الأعداد الصحيحه (integer) أي أن القيمه التي يحملها دائما ستكون عدد صحيح.

ثانيا / أن تكون لهذا المتغير أو الثابت قيمه عند أول أستخدام له داخل البرنامج فمثلا أننا عرفنا المتغير (x) لكن كم هي قيمة هذا المتغير هو عدد صحيح لكن كم !! فعندما نعطي الأمر (write (x)) فكم يجب على المترجم أن يطبع على شاشة التنفيذ !! لذا يجب أن نحدد قيمة المتغير أو الثابت قبل أول أستخدام .

هذه القيمه التي تحدد للمتغير تأتي من أحدى عمليتين فأما أن تأتي من داخل البرنامج أو تأتي من خارج البرنامج ... لنناقش الحالتين :

- **من داخل البرنامج :**

ويتم ذلك من خلال المعادلات أو التعابير الرياضيه أي بأستخدام المساواة (assignment) , فمثلا نقول

x : = 5 ;

هنا أستخدمنا المساواة والتي يرمز لها بلغة البرمجه باسكال بالشكل التالي (= :) وبذلك فان قيمة المتغير (x) ستكون مساويه الى العدد الصحيح (5) , أو ممكن أن تكون المعادله على شكل

x : = 3 * 2 +5 ;

هنا قيمة (x) تساوي (11) , وكذلك ممكن أن تحدد قيمه للمتغير بالمساواة ولكن في حقل الأعلان عن الثوابت.

ملاحظه : //

دائما عند وجود علامة المساواة (= :) فان الضوابط التاليه ستطبق :

- **يجب أن يكون هناك طرفين تفصل بينهما علامة المساواة , وبذلك ممكن أن نطلق عليها تسمية المعادله .**
- **الطرف الأيسر من المعادله أي الذي يقع على الجانب الأيسر من المساواة يكون متغير ومتغير واحد فقط دائما ولا يجوز أن يكون قيمه ثابته (مثلا 6 أو 456 أو 34.2 ..الخ) وكذلك لا يجوز أن يكون رمز معرف في حقل الأعلان عن الثوابت . كذلك لا يجوز أن يحتوي على علاقات رياضيه (مثل x + 6) .**
- **أما الطرف الأيمن فيمكن أن يكون قيمه عدديه واحده أو علاقه رياضيه تحتوي على قيم عدديه أو علاقه رياضيه تحتوي متغير واحد أو متغيرات أو متغيرات وقيم عدديه . مثلا**

```
X : = 89 ;
X : = 34 – 45 +3;
X : = y ;
X : = 3 * y +90 ;
```

- **عند تنفيذ البرنامج فأن المترجم سيبدأ بالطرف الأيمن من المعادله دائما ويتم فحص هذا الطرف فأذا كانت فيه متغيرات فسيبحث المترجم في الخطوات السابقه للتأكد من أن المتغير معرف أولا ثم يجب أن تكون له قيمه قبل هذه الخطوه وتجلب لتعوض عن المتغير في المعادله (ممكن أن نتخيل الطرف الأيمن عندها سيصبح عباره عن مجموعه من الثوابت) , بعدها تجرى العمليات الحسابيه وتكون من اليسار ألى اليمين وحسب أسبقيات العلامات الرياضيه المبينه في أدناه , فالأسبقيه الأعلى تنفذ أولا وأذا تساوت عمليتان بالأسبقيه فتنفذ العمليه التى في اليسار. من ذلك سينتج لنا عدد واحد بغض النظر عن قيمته هذه القيمه ستؤول الى المتغير الذي في الطرف الأيسر (دائما القيمه تنتقل من الطرف الأيمن للمعادله الى المتغير الذي في الطرف الأيسر) .**
- **يجب أن يكون المتغير الذي على يسار المساواة والمتغير أو المتغيرات على يمين المساواة من نفس النوع .**

أسبقيات العمليات الرياضيه Operator Precedence

تنفذ العمليات الرياضيه وفقا لأسبقياتها المبينه أدناه , دونت العمليات بشكل تنازلى أي من العمليه ذات الأسبقيه العليا الى العمليه ذات الأسبقيه الدنيا :

1. **– unary , Not**
2. *** , / , Mod , Div , And , Shl , Shr**
3. **+ , - , Or , Xor**
4. **< , <= , > , >= , <> , = , In**

ملاحظه ://

- **المعاملات بين أثنين من العمليات المختلفة الأسبقيه تنفذ حسب (operands) الأسبقيه (أي العمليه ذات الأسبقيه العليا أولا) .**
- **المعاملات بين أثنين من العمليات المتساوية الأسبقيه تنفذ من اليسار الى اليمين (أي العمليه التي في اليسار أولا) .**
- **الأقواس تنفذ أولا (أي أنها تأخذ أعلى أسبقيه) .**

- **<u>من خارج البرنامج :</u>**

وتتم عملية أسناد (أدخال) قيمه للمتغير أثناء تنفيذ البرنامج وذلك بأستخدام أمر القراءه (أقرأ) (read( ) , OR readln( )) وهي تعني (أقرأ القيمه المطبوعه على شاشة التنفيذ وحملها في الموقع الذي يشار اليه بواسطة المتغير الموجود بين القوسين) , **مثال**

```
Program CH2_Program2;
Var
  Num1, Num2, Sum: Integer;

Begin {no semicolon}
  Write ('Input number 1 :');
  Readln (Num1);
  Writeln ('Input number 2 :');
  Readln (Num2);
  Sum: = Num1 + Num2; {addition}
  Writeln (Sum);
  Readln;
End.
```

مخرجات البرنامج :

```
Input number 1:   20      { Press enter }
Input number 2:   15      { Press enter }
 35
```

شرح البرنامج : / /

أولا :// تم أستخدام حقل الأعلان عن المتغيرات للأعلان عن المتغيرات التي ستستخدم في البرنامج وهي (num1 , num2 , sum) وهي جميعا من نوع الأعداد الصحيحه لأن هذا البرنامج صمم للتعامل مع الأعداد الصحيحه (يقوم بجمع عددين صحيحين وأظهار النتيجة) .

ثانيا :// يمكن الأعلان عن كل متغير بسطر منفصل , ويمكن وضعها جميعا بسطر واحد كما في هذا البرنامج على شرط أن تكون جميع المتغيرات من نفس النوع (هنا جميعها أعداد صحيحه) وذلك لغرض تقليل المساحه التي يكتب عليها البرنامج , ويتم الفصل بين متغير وأخر **بفارزه** . وطبعا العباره تنتهي بفارزه منقوطه .

ثالثا :// بعد كلمة (begin) نلاحظ العباره التاليه ({ no semicolon }) وهي تعني لا تستخدم فارزه منقوطه , وبما أنها وضعت بين قوسين متوسطين فأن ذلك يعني أنها ملاحظه أو تعليق (Comment) للمستخدم أو القاريء بعدم أستخدام الفارزه المنقوطه بعد كلمة (begin) هذه العباره التي أعتبرت تعليق كتبت ووضعت بين قوسين متوسطين ({ }) , وبما أنها تعليق فيجب ان لا يكون لها تأثير على تنفيذ البرنامج (أي أنها تهمل أثناء تنفيذ البرنامج) , عليه فسيكون لنا قاعده أن أي عباره لغرض التوضيح أو التعليق ممكن كتابتها داخل البرنامج على أن تحاط بقوسين متوسطين وسوف لا تكون جزء من البرنامج أثناء التنفيذ (تهمل) .

ملاحظه ://

التعليقات أو الملاحظات تستخدم لأيضاح عمل بعض الدوال والأجراءات التي تكون معروفه لدى المبرمج وغير معروفه للمستخدمين , أيضا تستخدم لكتابة بعض المعلومات حول البرنامج (كوقت انشاءه أو تحديثه) أو معلومات حول المبرمج نفسه (مثلا الأسم , العنوان الألكتروني) .
التعليقات ممكن أن توضع في أي مكان في برنامج باسكال , ولكن يفضل أن تكتب في بداية البرنامج (في حالة كون المعلومات عن وظيفة البرنامج أو معلومات عن المبرمج) , أو تكتب بجانب الأوامر التي تحتاج الى توضيح . وممكن أن نضع التعليقات بين قوسين متوسطين ({ comments }) أو بين قوسين عاديين مع نجمه بجانب كل قوس ((* comments *)) ولا فرق بين الأثنين .

رابعا :// كما سبق وأن ذكرنا أن تنفيذ البرنامج يتم بالتسلسل من الأعلى الى الأسفل فيبدأ من كلمة برنامج ثم قسم الأعلان وقراءة المتغيرات بعدها ينفذ الأمر أبدأ ثم العبارة التي تليها وهي هنا الأمر أكتب (لاحظ الموجود بين القوسين في الأمر أكتب هو محصور بين علامتي أقتباس لذا فأنه يطبع كما هو) هذه العباره ستظهر على شاشة التنفيذ وهي تخبر المستخدم مايلي (أدخل الرقم الأول) وهي بشكل عام يمكن الأستغناء عنها دون أن يتأثر البرنامج .. ولكنها مفيده حيث تعلم المستخدم عن الخطوة أو الخطوات الواجب أتباعها لأنجاز تنفيذ البرنامج , (يمكن ملاحظة مثل ذلك في البرامج التي تعملون عليها مثلا في برنامج للعبه (game) معينه فأن هناك ملاحظات ستظهر على الشاشه لأرشاد المستخدم عن الخطوات الواجب أتباعها لتشغيل اللعبه أو أختيار درجة الصعوبه وغيرها).

خامسا :// هنا تبدأ عملية أدخال قيمه للمتغير (num1) وذلك بأستخدام الأمر (readln) , عند الوصول الى هذه الخطوه فأن شاشة التنفيذ (الشاشه السوداء) ستظهر ويكون هناك مؤشر صغير على شكل شارحه (-) يظهر ويختفي في موقع على الجانب الأيسر من شاشة التنفيذ , هذا المؤشر يخبر المستخدم بأن عليه أدخال قيمه (طباعة قيمه معينه بأستخدام لوحة المفاتيح) , وبعد أن

نطبع هذه القيمه يتم أعلام البرنامج (المترجم) بأنجاز العمل وذلك من خلال الضغط على الزر (Enter) في هذه الحاله سيتم قراءة القيمه التي طبعت على الشاشه وخزنها في الموقع الذي يؤشر عليه المتغير الموجود بين القوسين بعد الأمر(readln) وبذلك نكون قد أدخلنا قيمه للمتغير (num1) (خزنا قيمه) في الموقع الذي يؤشر عليه المتغير بعد هذه الخطوه , وهذا ما أسميه الأدخال من خارج البرنامج (أي الأدخال الذي يتم بواسطة المستخدم أثناء تنفيذ البرنامج).

ملاحظه ://

الفرق بين الأمر Read والأمر Readln وكلاهما يستخدم للقراءه , أن الأمر Read يقرأ المدخلات التي يدخلها المستخدم ويتوقف بعد أخر قيمه يتم قرائتها فأذا جاء أمر (read) أخر فيبدأ القراءه أعتبارا من الموقع الذي يؤشر عليه بواسطة المؤشر بعد القراءه الأولى .
أما الأمر Readln فأنه يعمل بنفس الطريقه مع أستثناء واحد . فبعد أن ينتهي من قراءة أخر قيمه في القائمه فأن المؤشر سيهمل كل القيم المتبقيه على ذلك السطر وينتقل الى بداية سطر جديد .

سادسا :// الأمران اللاحقان هما مشابهان للخطوتين الرابعه والخامسه.

سابعا :// المعادله (sum : = num1 + num2) عند الوصول الى هذه المعادله فأن المترجم سيبدأ بالطرف الأيمن من المعادله ويعوض عن المتغيرات الموجوده بما يساويها من قيم ثم أجراء عملية الجمع على هذه القيم لينتج عن ذلك قيمه واحده في الطرف الأيمن , هذه القيمه ستوضع (تخزن) في الموقع الذي يؤشر عليه المتغير الموجود في الطرف الأيسر, وبذلك فان المتغير (sum) ستسند له قيمه (تخزن في الموقع الذي يؤشر عليه) من خلال المعادله وهذا ما أسميه أدخال قيمه من داخل البرنامج (أي أن المستخدم لا يتدخل في ذلك أثناء تنفيذ البرنامج) .

ثامنا :// بعد أنجاز العمل المطلوب من البرنامج فلا بد من أعلام المستخدم بالنتيجه المتحصله من أنجاز أو تنفيذ هذا البرنامج , ويتم ذلك من خلال طباعة القيمه المتحصله والتي هي الأن موجوده في المتغير (sum) , لذا تم أستخدام الأمر أكتب ليطبع ما موجود بين القوسين اللتين بعده , ولما كان ما موجود ضمن القوسين غير محدد بعلامتي أقتباس لذا فان القيمه التي يحملها هذا المتغير هي التي تظهر على شاشه التنفيذ .

تاسعا :// سبق وأن نبهنا ألى أن سرعة التنفيذ سوف لاتمكن المستخدم من رؤيه النتائج وبينا بأن من الممكن أن نستخدم الزرين (Alt + F5) معا لأظهار النتائج ... وممكن أيضا أن نستخدم طريقه أخرى وهي أستخدام الأمر (readln) فقط أي بدون الأقواس التي بعده كما مبين أعلاه , هنا المترجم سيفسر الأمر كما لو أن المستخدم يرغب بادخال قيمه لمتغير وسيظهر شاشة التنفيذ منتظرا المستخدم ليدخل القيمه وبذلك سنرى النتائج ويمكن الخروج من هذه الشاشه وذلك بالضغط على الزر (Enter) . (هي عميلة أحتيال على الحاسوب وعند الضغط على الزر Enter فهذا يعني أن عملية

الأدخال تمت والمفروض أن يقرأ ما مطبوع على الشاشه ويسنده الى المتغير والذي هو غير موجود) .

عاشرا :// الأمر الأخير هو (end) وواضح أنه يعني النهايه (هنا يجب أن نلاحظ أننا يمكن أن نستخدم الأمر (begin) أكثر من مره بشرط أن يكون هناك أمر (end) لكل أمر (begin) , جميع هذه الأوامر (end) تنتهي بفارزه منقوطه عدا أخر أمر (end) فأنه سينتهي بنقطه) , ان الأمر (end) الذي ينتهي بنقطه يعني نهاية البرنامج ككل وكل مابعد هذا الأمر يهمل وكما هو واضح أعلاه .

ملاحظه ://

أدناه بعض القواعد التي يجب أن تلاحظ عند أدخال البيانات المطلوبه :

1. **يجب أن يتطابق عدد البيانات التي يتم أدخالها مع عدد المتغيرات المدونه بين القوسين في أيعاز القراءه .**
2. **يجب أن يتطابق نوع القيمه المدخله لمتغير معين مع النوع المعلن لهذا المتغير .**
3. **المتغيرات المدونه بين القوسين في أيعاز القراءه يجب أن تفصل بينها فارزه أذا كان عددها أكثر من متغير واحد .**
4. **أذا كان أكثر من متغير واحد في أيعاز قراءه واحد فيمكن أدخالها جميعا ثم ضغط الزر (Enter) على أن يفصل بين قيمه وأخرى فراغ , أو ندخل القيم واحده بعد الأخرى على أن نضغط الزر (Enter) بعد أدخال كل قيمه .**
5. **لا يجوز أن تكون القيم المدخله صيغ رياضيه (أي قيم بينها علامات رياضيه)**

2.4 متغيرات السلاسل الحرفيه STRING VARIABLES

الأن نتعلم كيفيه أدخال نص كتابي من قبل المستخدم وذلك بأستخدام متغيرات السلاسل الحرفيه و البرنامج التالي يبين كيفيه أدخال متغيرات السلاسل الحرفيه وذلك بتحفيز المستخدم لهذا الاجراء , **مثال**

```
Program CH2_Program3;
Var name, surname: String;

Begin
  Write ('Enter your name :');
  readln (name);
  Write ('Enter your surname :');
  readln (surname);
  writeln ;{new line}
  writeln ;{new line}
  Writeln ('your full name is: ', name,' ', surname);
  Readln;
End.
```

مخرجات البرنامج :

```
Enter your name:      Ahmed     { enter  }
Enter your surname:   Abass     { enter  }

Your full name is: Ahmed  Abass
```

شرح البرنامج : //

اذا ما نظرنا الى هذا البرنامج فمن الممكن أن نسجل عدد من الملاحظات ونتعلم أشياء جديده وهي :

اولا :// تم أستخدام المتغيرين (name , surname) من نوع سلاسل حرفيه (string) ولذلك فعند تنفيذ البرنامج سيتم الطلب من المستخدم أدخال مجموعه من الحروف لكل من المتغيرين وما يتم طباعته أو أدخاله عن طريق لوحة المفاتيح سيذهب مباشرة الى الذاكره ويخزن بالموقع المؤشر عليه بواسطة المتغير أعلاه . أن هذين المتغيرين يمكن استبدالهما بأي متغير أخر مثلا (n) بدلا من (name) وطبعا ذلك لا يؤثر في البرنامج كما سبق وأن بينا.

ثانيا :// لاحظ في السطرين التاسع والعاشر أستخدام الأمر (writeln) دون وجود أقواس بعدهما وهذه العمليه تفيد بطباعة سطر فارغ لكل منهما أي أن المؤشرسيتحرك الى سطر جديد في كل مره يرد مثل هذا الأمر, وهي لأغراض تنظيم وترتيب المخرجات .

ثالثا :// أمر الطباعه الأخير هو أيضا يختلف عن ذلك الذي سبق وأن تم أستخدامه , حيث أنه يحتوي على عدد من العبارات داخل القوسين يفصل بين واحده واخرى فارزه , وهذه طريقه يمكن أستخدامها مع أوامر الطباعه وذلك بدلا من تكرار أمر الطباعه لكل حاله فيمكن دمجها جميعا بأمر واحد على أن يفصل بين واحده وأخرى فارزه.
لاحظ الجزء الأول وضع بين علامتي أقتباس (:Your full name is) وهذا الجزء سيتم طباعته كما هو كما تعلمنا , أن الفارزه تعني البدأ بأمر (write) جديد لما يلي الفارزه أي وكأنما نقول (write (name)) , ولذا فهي ستطبع مايلي الفارزه وعلى نفس السطر ولما كان المتغير (name) ليس محدد بعلامتي أقتباس فسيتم طباعة مايحمله من قيمه , والقيمه التي يحملها هنا هي عباره عن سلسله حرفيه . بعدها نلاحظ الفارزه ثم تليها علامتي أقتباس تحدد بينها فراغ لذا فان أمر الطباعه يشير الى طباعة فراغ وهكذا سيتم طباعة فراغ حسب حجم الفراغ المحدد بين علامتي الأقتباس وأخيرا يأتي أمر الطباعه للمتغير (surname) وسيتم طباعة قيمته والتي هي سلسله حرفيه.
ممكن أعادة كتابه البرنامج (3) ولكن بأستخدام الثوايت لنلاحظ كيفية أستخدام الثوابت , جزء من البرنامج ممكن أن يكون حقل الثوابت , وهذا يختلف عن المتغيرات حيث أن المعرف في هذا الحقل تبقى قيمته ثابته في البرنامج ولا يمكن تغييرها , هذا البرنامج لا يختلف كثيرا عن البرنامج السابق ولكن تم أستخدام المعرف (name) كقيمه ثابته ولذلك سوف لا نحتاج الى أدخال قيمه لهذا المتغير أو المعرف لأننا أدخلنا له قيمه من داخل البرنامج وممكن أن تكون هذه القيمه عدد , حرف , أو سلسله حرفيه .

```
Program CH2_Program4;
Const          {the reserved word 'const'
                is used to initialize constants}
  Name = 'Ahmed';
Var
  Surname: String;
Begin
  Write ('Enter your surname :');
  Readln (surname);
  Writeln;
  Writeln;
  Writeln ('your full name is: ', name,' ', surname);
  Readln;
End.
```

مخرجات هذا البرنامج هي ذات المخرجات للبرنامج السابق .

ملاحظه ://:

يستخدم الأمر (Halt) لأيقاف تنفيذ البرنامج .
ويستخدم الأمر (Exit) للخروج من الكتله الحاليه .

2.5 أنواع الأخطاء التي تحدث في البرنامج

هناك ثلاث أنواع من أخطاء البرامج المهمه وهي :

2.5.1 الخطأ الذي يمكن تميزه أثناء ترجمة البرنامج قبل التنفيذ , **مثال** عدم كتابة الأمر (End) لأمر (begin) موجود في البرنامج . وهذا نوع من الأخطاء التي تسمى أخطاء وقت الترجمه (**Compile _ time errors**) , وهو سيؤدي الى عدم تنفيذ البرنامج مع ظهور رسالة خطأ تحدد نوع الخطأ .

2.5.2 الخطأ الذي لا يميز أثناء الترجمه وأنما يميز أثناء التنفيذ , **مثال** أثناء التنفيذ كان هناك أمر لأيجاد الجذر التربيعي لمتغير معين وظهر أن قيمة المتغير سالبه (معروف أنه لا يمكن أيجاد الجذر للقيم السالبه) , هذا سيؤدي الى توقف تنفيذ البرنامج مع ظهور رسالة خطأ . هذا النوع من الأخطاء التي تسمى أخطاء وقت التنفيذ (**Run _ time errors**) .

2.5.3 الأخطاء التي لا تميز سواء أثناء الترجمه أو أثناء وقت التنفيذ , **مثال** الأخطاء التي يرتكبها المبرمج ربما سهوا ولكنها مقبوله للحاسوب كأن يكون المطلوب أدخال القيمه (30) ويتم أدخال القيمه (40) هذا الخطأ لا يعترض عليه الحاسوب ولكنه سيؤدي الى ظهور نتائج غير صحيحه , مثل هذه الأخطاء سوف تسمح للبرنامج بالأستمرار بالتنفيذ .

ملاحظه ://

عند حدوث خطأ من النوعين الأول والثاني أعلاه فأن البرنامج سوف لا ينفذ وتظهر رسالة خطأ يمكن أن تضغط الزر (F1) للحصول على مساعده توضح ماهية الخطأ , كذلك فأن مؤشر على شكل شريط أحمر يظهر في الأعلى يبين رقم الخطأ , ومؤشر على شكل شارحه يؤشر على موقع الخطأ أو السطر الذي بعده أحيانا .

جدول (2.1) : الرموز الخاصه بلغة البرمجه باسكال

تستخدم لجمع قيمتين , تستخدم أيضا لأتحاد سلسلتين حرفيتين , وتستخدم لجمع مجموعتين	(PLUS) علامة الجمع	+
تستخدم لطرح قيمتين , تسبق القيم للدلاله على أن القيم سالبه , وتستخدم للفرق بين مجموعتين	(MINUS) علامة الطرح	-
تستخدم لأجراء عملية الضرب بين قيمتين , وكذلك لتقاطع مجموعتين	(ASTERISK) علامة الضرب	*
تستخدم لأجراء عملية القسمه بين قيمتين وتكون النتيجه قيمه كسريه	(SLASH) علامة القسمه	/
تستخدم لفحص المساواة بين قيمتين	(EQUAL) علامة المساواة	=
تستخدم لمقارنة قيمتين وتحديد القيمه الأصغر	(LESS THAN) أقل من	<
تستخدم لمقارنة قيمتين وتحديد القيمه الأكبر	(GREATER THAN) أكبر من	>
يستخدم في المجاميع والمصفوفات (بالأشتراك مع القوس الأيمن)	(LEFT BRACKET) القوس المربع الأيسر	[
يستخدم في المجاميع والمصفوفات (بالأشتراك مع القوس الأيسر)	(RIGHT BRACKET) القوس المربع الأيمن	]
تستخدم في أختيار حقل لمتغير القيود , وتأتي بعد الأمر (end) لأنهاء البرنامج	(PERIOD) النقطه	.
تستخدم لفصل العوامل (arguments) ولفصل المتغيرات في حقل الأعلان عن المتغيرات ولفصل مديات المصفوفه الثنائيه	(COMMA) الفارزه	,
تستخدمان لفصل أنواع المتغيرات في حقل الأعلان عن المتغيرات ولفصل أسم الداله عن نوعها عند تعريفها	(COLON) النقطتان المتعامدتان	:

;	(SEMI-COLON) الفارزه المنقوطه	تستخدم لفصل العبارات في لغة البرمجه باسكال
^	(POINTER) المؤشر	تستخدم لتعريف الأنواع والمتغيرات من نوع مؤشر وتستخدم للوصول الى محتويات المتغيرات من نوع مؤشر وكذلك محتويات الفايلات
(	(LEFT PARENTHESIS) القوس الأعتيادي الأيسر	يستخدم ليحتوي التعابير الرياضيه والمنطقيه أو معاملات الدوال والأجراءات (بالأشتراك مع القوس الأيمن)
)	(RIGHT PARENTHESIS) القوس الأعتيادي الأيمن	يستخدم ليحتوي التعابير الرياضيه والمنطقيه أو معاملات الدوال والروتينات الفرعيه (بالأشتراك مع القوس الأيسر)
< >	(LESS THAN / GREATER THAN) عدم المساواة	أختبار عدم المساواة بين قيمتين
< =	(LESS THAN / EQUAL) أصغر من أو يساوي	للمقارنه بين مجموعتين , و أختبار المجموعه الجزئيه
> =	(GREATER THAN / EQUAL) أكبر من أو يساوي	للمقارنه بين قيمتين , وأختبار المجموعه الشامله (superset)
: =	(COLON / EQUAL) المساواة	تستخدم لأسناد قيم للمتغيرات
..	(PERIOD / PERIOD) النقطتان المتجاورتان	تفصل

جدول (2.2) : الكلمات المحجوزه الخاصه بلغة البرمجه باسكال

AND	معامل الربط المنطقي
ARRAY	نوع المصفوفه
BEGIN	عبارة البدأ
CASE	عبارة البدأ (في حالة) (case)
CONST	تعريف الثوابت
DIV	تقسيم الأعداد الصحيحه ونتيجتها أيضا أعداد صحيحه
DO	تتبع أوامر التكرار (For , While) وتسبق الافعال التي يجب أن تنفذ
DOWNTO	تستخدم مع حلقة التكرار (For) وتدل على أن المتغير يقل في كل

	دوره
ELSE	تستخدم مع (If) , حيث في حالة كون التعبير المنطقي بعد (if) خطأ (false) تنفذ العباره التي بعد (else)
END	نهاية العبارات المركبه التي تبدأ بالأمر ابدأ (begin) , أو نهايه الأمر (case) أو نهايه تعريف القيود (record)
FILE	تعرف متغير من نوع ملف
FOR	تنفذ سطر أو أكثر من الأوامر بشكل متكرر طالما المتغير لم يصل الى نهايته
FUNCTION	تعرف دالة باسكال
GOTO	تفيد للتفرع الى عنوان محدد (label)
IF	تفحص شرط منطقي وتنفذ العباره أذا كان الشرط صح (true)
IN	تحدد التعبير المنطقي على أنه صح (true) أذا كانت القيمه المفحوصه ضمن مجموعه خاصه
LABEL	تحدد العنوان الذي سيتم التفرع له عند استخدام الأمر (Goto)
MOD	لحساب باقي قسمة عددين
NIL	القيمه الخاليه للمؤشرات (pointers)
NOT	لنفي قيم العبارات المنطقيه
OF	تستخدم في عبارة (case) بعد متغير (case)
OR	معامل الأختيار المنطقي (أو)
PACKED	يستخدم مع المصفوفات , الملفات , القيود , والمجموعات لضغط البيانات المخزنه
PROCEDURE	تعرف الروتين الفرعي في باسكال
PROGRAM	يستخدم في بداية البرنامج (رأس البرنامج)
RECORD	للتصريح عن متغير من نوع القيود
REPEAT	للبدأ بأمر التكرار (repeat)
SET	لتعريف المجموعات
THEN	تلي التعبير المنطقي بعبارة (IF)
TO	تستخدم في حلقة التكرار (for) للدلاله على أن المتغير يزداد في كل دوره بمقدار واحد
TYPE	للتصرسح عن نوع جديد من المتغيرات
UNTIL	يستخدم لأنهاء حلقة التكرار (repeat)
VAR	للتصريح عن متغيرات البرنامج
WHILE	ينفذ كتله من الأوامر بشكل متكرر لحين أن يصبح الشرط (False)
WITH	لتحديد متغير قيد يستخدم مع كتله من الأوامر

2.6 أمثله محلوله

- أكتب برنامج لتحويل (42200 sec) الى ما يقابلها بالساعات و الدقائق و الثواني .

```
Program CH2_Program5;
Var
  Hour, min, sec, temp: integer;
Begin
  Sec:=42200 mod 60;
  Temp:=42200 div 60;
  Min:=temp mod 60;
  Hour: = temp div 60;
  Writeln ('hour=', hour,'min=', min,'sec=', sec);
End.
```

- أكتب برنامج لأيجاد قيمة (y) من المعادله $y = 4x^2 + 3x - 6$

```
Program CH2_Program6;
Var
  X,y:integer;
Begin
  X: =6;
  Y: =4*sqr(x) +3*x-6;
  Writeln(y);
End.
```

- أكتب برنامج لتحويل درجة حراره مقاسه بالفهرنهايت الى درجه مئويه .

```
Program CH2_Program7;
Var
  F:integer;
  C: real;
Begin
  Writeln ('Enter temerature degree in fehrnhite ');
  Readln (f);
  C :=( 5/9)*(f+32);
  Writeln(c);
End.
```

- أكتب برنامج لأيجاد مساحة ومحيط دائره .

```
Program CH2_Program8;
Var
  R: integer;
  Area, perimeter: real;
Begin
  writeln ('Enter circle radius');
  Readln(r);
  Area: =sqr(r)*pi;
  Perimeter: =2*r*pi;
  writeln ('area= ',area,'perimeter= ',perimeter);
End.
```

- أكتب برنامج لأيجاد حاصل ضرب ومعدل ثلاث أرقام .

```
Program CH2_Program9;
Var
  Prod, a, b, c: integer;
  Average: real;
Begin
  Writeln ('Enter three numbers');
  Readln (a, b, c);
  Prod: = a*b*c;
  Average :=( a + b + c)/3;
  Writeln ('prod= ', prod);
  Writeln ('averge= ', average);
End.
```

الفصل الثالث
أيعازات القرار والتكرار
DECISION AND REPEAT INSTRUCTIONS

3.1 المقدمه

الأن جاء دور دراسة القواعد الأكثر أهميه في البرمجه . وهي أيعازات القرار (If statement) وكذلك الأيعاز المراغف لها (Else) وعبارات التكرار والتي هي (For loops , Repeat .. Until loop, While loop) . غالبا تعتبر هذه الأوامر من الأوامر الكثيرة الأستخدام في البرمجه لذا ننصح بعد الأنتهاء من دراسة هذا الفصل الشروع بكتابة برامج تستخدم فيها هذه القواعد وزيادة الخبره العمليه قبل الأنتقال الى موضوع جديد.

3.2 عبارة أذا IF STATEMENT

يستخدم هذا الأمر بالترافق مع (then) كما سنوضح وهو يفيد لأتخاذ قرار من قبل المترجم بناءا على بعض المعطيات التي ترد في البرنامج , هناك العديد من الحالات التي لايمكن التنبأ بها من قبل المستخدم أثناء كتابة البرنامج , فعلى سبيل المثال أننا نكتب برنامج لأيجاد الجذر التربيعي لأعداد صحيحه يتم أدخالها من قبل المستخدم أثناء تنفيذ البرنامج , في هذه الحاله وكما معلوم فأن العدد الصحيح يجب أن يكون موجب لأنه لايمكن أيجاد الجذر التربيعي للعدد السالب , السؤال هنا هل يمكن منع المستخدم من أدخال عدد سالب سواء كان بقصد أو سهوا , أن المبرمج سوف لايجد وسيله أثناء كتابة البرنامج لمعالجة هذا الأشكال البسيط ألا أن يستخدم عبارة القرار(أذا) والتي ممكن أن تكون كما يلي (أذا كان العدد موجب أوجد الجذر التربيعي وبخلاف ذلك أهمله) .. (وبالتأكيد فأن المترجم في الحاسوب لا يفهم عبارة موجب لذا نستبدلها بما يتناسب وقواعد لغة البرمجه باسكال فنقول أذا كان العدد أكبر أو يساوي صفر فأوجد الجذر التربيعي).
أن أستخدام عبارة (If) يكون كما يلي (أذا (شرط) عليه) .. (if condition then) أذا تحقق الشرط الذي بعد الأمر (If) فيتم تنفيذ العباره التي بعد (then) أما أذا لم يتحقق هذا الشرط فيهمل ما بعد (then) أذن ستكون طريقة كتابه هذا الأمر كما يلي :

If conditional expression true **then** code ... ;*{if one action }*

ملاحظه ://
لا توجد بعد الأمر (If) أو (then) فارزه منقوطه .

ممكن مثلا أن نطلب من أحدهم عملا ونقول له (أذا كان المحل مفتوحا فأجلب لي شراب الببسي) (If shop open then get me pepsi) هذه العباره ممكن صياغتها برمجيا كما يلي :

```
If shopopen then
    Drink : = pepsi ;
```

نلاحظ في هذا المثال أن الفعل المطلوب أنجازه هو (واحد أن يجلب لنا شراب الببسي) .
أما أذا كان ما مطلوب أنجازه هو أكثر من فعل واحد فأن الصيغه ستختلف قليلا :

```
If conditional expression true  then

Begin

instructions ...

End;            {if more than one action is required}
```

ماذا يعني ذلك أن الأمر (If) ينفذ عباره واحده تأتي بعد (then) والتي تمثل الفعل المطلوب أنجازه عند تحقق الشرط , أما أذا كان هناك أكثر من فعل واحد مطلوب أنجازه عند تحقق الشرط فيجب أن نحدد هذه الأفعال للمترجم ويكون ذلك بأن نحددها بين الأمرين ((begin) و (end)) وبذلك سيكون واضح أن الأفعال المطلوب تنفيذها عند تحقق الشرط تبدأ بعد الأمر (begin) وتنتهي بالعباره التي قبل (end) مع ملاحظة أن (end) تنتهي **بفارزه منقوطه**.
لنعد الى المثال السابق ونطلب من أحدهم عملا ونقول (أذا كان المحل مفتوح فأجلب لي شراب الببسي وعلبة سكائر كنت) (if shop open get me pepsi and kent cigrates)
الفعل المطلوب أنجازه هنا هو أكثر من واحد حيث المطلوب جلب شراب الببسي وسكائر من نوع كنت . لذا ستكون صياغة هذه العباره برمجيا كما يلي :

```
If shopopen then
    Begin
       Drink   : = pepsi ;
       Smook  : = kent ;
   End ;
```

في حالة عدم وضع (end , begin) فان أول عباره ستأتي بعد (then) هي التي ستعامل على أنها تعود الى الأمر (If) وتنفذ في حالة تحقق الشرط وهي هنا ستكون (drink) أما أستخدام (begin , end) فهي دلاله للمترجم على أن مجموعة الأيعازات المحصوره بين (begin , end) هي جميعا مطلوب تنفيذها أذا ما تحقق الشرط .

ملاحظه ://

عند الحاجه لأستخدام المساواة في الشرط بعد (If) فلا تستخدم المساواة المسبوقه بنقطتين (= :) (assignment) وأنما تستخدم المساواة الأعتياديه (=) لأن أستخدام الأولى سيؤدي الى عدم أكمال التنفيذ وظهور رسالة خطأ .

هناك حاله أخرى عند أستخدام (If) , هو أستخدامها لأختيار فعل واحد من أثنين فمثلا في مثالنا السابق ممكن أن يكون الطلب كما يلي (أذا كان المحل مفتوحا فأجلب لي شراب الببسي وبخلاف ذلك (أي أذا كان المحل مغلق) فأعمل لي قهوه) (If shop open then get me pepsi otherwise get me a coffee) هذه العباره تنفذ برمجيا كما يلي :

```
If shopopen then
    Drink : = pepsi
 Else
    Drink : = coffee ;
```

لنلاحظ هنا أن الشرط الذي بعد (If) أما أن يكون (صح , أو خطأ) (true OR false) أي أما أن يكون المحل مفتوح أو مغلق ولا يوجد أحتمال أخر. فأذا كان المحل مفتوح فالمطلوب أن يجلب شراب وهو البيسي في خلاف ذلك (else) أي أذا كان المحل مغلق فليكن الشراب هو قهوه . الملاحظه المهمه هنا هي أنه لايمكن أن ينفذ العملان سوية أي لا يمكن أن يجلب ببسي وقهوه في نفس الوقت والسبب هو أنه لايمكن أن يكون المحل مفتوح ومغلق بذات الوقت . عليه فأذا تحقق الشرط (أي الشرط صح بمعنى أن المحل مفتوح) فأن العباره التي تأتي بعد (then) ستنفذ بينما العباره التي بعد (else) ستهمل , أما أذا كان الشرط غير متحقق (أي أجابة الشرط خطأ بمعنى أن المحل مغلق) فأن العباره التي بعد (If) ستهمل وتنفذ العباره التي بعد (else) .

ملاحظه ://

دائما العباره التي تسبق (else) مباشرة لا تحتوي على فارزه منقوطه (تحذف).

المثال التالي ممكن أن يكون جزء من لعبه بامكانك ان تضيف أليها أسئله أخرى لتكون لعبه متكامله :

```
Writeln ('Who has discovered the land of America?');
Readln (ans);
If (ans = 'Christopher Columbus') then
 score := score + 1                          {if this is false,}
ELSE
 Writeln ('sorry, you've got it wrong!');     {then this is true}
```

3.3 أذا المركبه COMPOUND IF

ممكن أن نستخدم (If) بشكل متداخل مع (If OR else) أخرى , وبهذه الحالة تسمى مركبه (أي ممكن أن يكون بعد (then) عباره (If) وممكن أيضا بعد عبارة (else) وممكن أن تكون أكثر من واحده . **فمثلا** نريد أن نفحص نوعية رمز معين ووفقا لذلك نقرر ماهو الأجراء الواجب أتباعه وكمايلي :

```
If ( charkind = digit ) then
     Readnumber
  Else
   If  ( charkind = letter ) then
        Readname
     Else
      Reporterror  ;
```

لنتأمل هذا المثال ففي البدايه يتم فحص الشرط لمعرفة نوع الرمز للمتغير (charkind) هل هو رقم (digit) أم لا , وكما تعلمنا دائما أن الأجابه أما نعم (صح) أو لا (خطأ) ولا يوجد أحتمال أخر , فأذا كان صح معناه أن الرمز من نوع (digit) عليه تنفذ العباره التي بعد (then) مباشرة أي أقرأ رقم (هذا الأحتمال الأول) , أما الأحتمال الثاني فتكون أجابة الشرط خطأ أي أن نوع الرمز هي ليست أرقام عليه فستهمل العباره التي بعد (then) وتنفذ العباره التي بعد (else) .. عندما يحين الدور لتنفيذ العباره التي بعد (else) نلاحظ أن هذه العباره هي أيضا عبارة (If) هذا يعني أنه لازال هناك أحتمالات أخرى يجب أن تفحص فممكن أن يكون الرمز ه و (letter) أو شيء أخر وتطبق نفس القاعده فأذا كانت أجابة الشرط صح تنفذ العباره التي بعد (then) (الثانيه) أما أذا كانت الأجابه خطأ فتنفذ العباره التي بعد (else) (الثانيه) والتي هي أصدار رسالة خطأ (أي أعلام المستخدم أن هذا الرمز هو ليس (digit OR letter) عبارات (If) هذه تسمى أيضا عبارات (If) المتداخله (nested If statements)

```
If (this happens) then           {if 1}
   If (this happens) then        {if 2}
      (do this) etc...
   Else (do this)                {if 2}
 Else (do this) etc...           {if 1}
```

ملاحظه ://

دائما تستخدم (If) عندما نحتاج أن نختار بين أكثر من حاله (أي أختيار عمل أو حاله واحده من بين أثنين أو أكثر) .

3.4 عبارة التكرار REPEAT -- UNTIL LOOP

يستخدم هذا الأمر لتكرار عباره أو أكثر لعدد من المرات وفقا لمتطلبات البرنامج والتي يحددها المبرمج , في هذا الأمر فان البرنامج سينفذ على الأقل لمره واحده .. ويكون توقف البرنامج أعتمادا على شرط يوضع بعد (until).
التكرار يبدأ بالأمر (أعد أو كرر) (Repeat) ثم مجموعه من الأيعازات المطلوب تكرارها وتنتهي بالأمر (لغاية) (until) الذي يكون بعده شرط (أي لغاية تحقق هذا الشرط) , المترجم حين يجد العباره (أعد) فأنه يعلم أن المطلوب أعادة تنفيذ العبارات المحصوره بين هذا الأمر والأمر (لغاية).. في كل مره يصل المترجم الى الأمر (لغاية until) يفحص الشرط الذي بعده فأذا كان الشرط غير متحقق (أجابته false) فأن المترجم سيعود الى الأمر (repeat) ويبدأ بالتنفيذ نزولا من جديد , هذه العمليه تستمر لغاية أن يتحقق الشرط وتكون أجابته (true) . الصيغه القواعديه لهذا الأيعاز هي :

```
Repeat
    Instruction 1 ;
   Instruction 2 ;
   Etc…
Until (condition is true ) ;
```

مثال : // هذا برنامج بسيط واجبه أدخال أسماء الطلبه وطباعتها , البرنامج لا يتوقف لغاية أدخال أسم (علي) .

```
Program CH3_Program1;

Var YN: String;
Begin
  Writeln ('enter name of students?');
  Readln (YN);
  If (YN <> ' Ali') then
    Writeln (YN);
  Readln (YN);
  If (YN <> ' Ali ') then
    Writeln (YN);
  Readln (YN);
  If   (YN <>  ' Ali ') then
    Writeln (YN);
  Readln (YN);
  If (YN <> ' Ali') then
    Writeln (YN);
  Readln (YN);
  If (YN <> ' Ali ') then
    Writeln (YN);
  Readln (YN);
  If   (YN <>  ' Ali ') then
    Writeln (YN)
     …
     …
```

هذا البرنامج ممكن أن يستمر بعدد كبير من الخطوات المتشابه وحسب عدد الطلبه المراد طباعة أسمائهم , أن العبارات (اقرأ , أذا , وأكتب) تتكرر بأستمرار في البرنامج أعلاه , لذا فان لغة البرمجه باسكال أوجدت البديل الذي يسهل العمل ويختصر عدد الخطوات ألا وهو عبارات التكرار . واحد من هذه الأوامر هو (repeat) وأذا ما أعدنا كتابة البرنامج أعلاه ولكن مع أستخدام (repeat) , سينتج لنا البرنامج التالي :

```
Program CH3_Program2;

Var YN: String;
Begin
  Writeln ('enter name of students?');
  Repeat        {repeat the code for at least one time}
    Readln ( YN );
    Writeln ( YN ) ;
  Until (YN = 'Ali');
End.
```

ميزة هذا الأمر أن الشرط هو في نهاية التكرار ولذا فأنه سينفذ ولو لمره واحده قبل أن يتم فحص الشرط . أرجو ملاحظة كيف أن البرنامج أصبح أكثر أختصارا وأسهل للمتابعة .

3.5 عبارة التكرار WHILE - DO LOOP

وهو أيضا من أيعازات التكرار وهو يشابه ألى درجه كبيره الأيعاز (Repeat_until) حيث أن واجب الأيعازين هو التكرار لمرات غير محدده ابتداءا وأنما يعتمدان على تحقق شرط معين لأيقاف التكرار , الصيغه القواعديه لهذا التكرار هي :

While *<condition is true>* ***do*** *the following:*

instruction 1;

instruction 2;

instruction 3;

etc...

End; {If while-do loop starts with a begin statement}

ماذا يعني هذا الأمر (عندما يتحقق الشرط أعمل ما يلي) وفي كل مره سينفذ الأيعاز الذي بعده مباشرة ويعود الى (while) ليفحص الشرط هل هو متحقق أم لا فأذا كان متحقق ينفذ وأن كان غير متحقق سيهمل الأيعاز الذي بعد (while) وينفذ ما بعده .

ملاحظه ://

كما هو الحال في (If and else) فان الأمر (while) ينفذ عباره واحده فقط والتي تأتي بعده مباشره , أما أذا كان هناك أكثر من عباره واحده مطلوب تكرارها ضمن الامر (while) فيجب أن تحدد بين الأمر (begin) والأمر (end)

اذن ما هو الفرق بين (While) و (Repeat) .. لاحظ الجدول (3.1) :

جدول (3.1) : الفارق بين أمري التكرار (repeat , while)

While_Do	Repeat_Until
الشرط في بداية التكرار	الشرط في نهاية التكرار
لا ينفذ أي أيعاز مالم يتم فحص الشرط وتحققه	سيتم تنفيذ الأيعازات المشموله بالتكرار على الاقل مره واحده قبل أن يتم فحص الشرط
تنفذ الأيعازات المشموله بالتكرار عند تحقق الشرط	يتوقف التنفيذ عند تحقق الشرط
يستخدم مع طلبات التكرار غير المحدده بعدد ثابت من التكرارات مسبقا	يستخدم مع طلبات التكرار غير المحدده بعدد ثابت من التكرارات مسبقا
يعتمد أستمرار التنفيذ على تحقق الشرط ويتوقف التنفيذ عند عدم تحقق الشرط	يعتمد أستمرار التنفيذ على عدم تحقق الشرط ويتوقف التنفيذ عند تحقق الشرط

مثال :// مثال بسيط لأدخال مجموعة أرقام وطباعتها بشرط يتم التوقف عند أدخال الرقم (0) .

```
Program CH3_Program3;

Var x : integer;
Begin
  Writeln ('Enter number');
  Readln(x);
  While x <> 0 do
    Begin
      Writeln(x);
      Readln(x);
    End;
End.
```

شرح البرنامج ://

المطلوب من البرنامج أدخال مجموعة أرقام بشرط أن يتوقف عند أدخال الرقم (0) , أذن لما كان أدخال مجموعة أرقام فهذا يعني أننا سنكرر أمر الأدخال أكثر من مره وفي كل مره يجب فحص الرقم لغرض طباعته أذا لم يكن يساوي (0) هذه العمليه ممكن تكرارها 5 مرات 10 مرات 1000 مره أو أكثر حسب طبيعة العمل (تصوروا برنامج يتكون من هذا الكم الهائل من الخطوات المتشابهه !!) لذلك لتجنب هذه العمليه تم أيجاد أيعازات التكرار فيمكن هنا أن نستخدم الأمر (While) لأختصار البرنامج , هذا الأمر يحتاج الى شرط لغرض العمل والتوقف , في هذا المثال البرنامج يتوقف عند ورود الرقم (0) أي أنه يعمل مع الأرقام الأخرى ولما كان الشرط يجب أن يكون (true) لكي يعمل أذن أي رقم لايساوي صفر سوف يجعل البرنامج يعمل لذا جعلنا (x < > 0) , لقد سبق وأن بينا أن المترجم عندما يصل الى أي خطوه فيها متغير سيقوم بعملين الأول يتأكد من تعريف المتغير في حقل المتغيرات والثاني يتأكد من أن المتغير له قيمه وحسب النوع المعلن عنه في حقل المتغيرات . لذا فأنه عندما يصل المترجم الى الأمر (While) يجب أن يجد قيمه للمتغير (x) وهذا هو السبب الذي جعلنا ندخل قيمه للمتغير (x) قبل الأمر (While) وأن لم نقم بذلك فأن البرنامج سيفشل لعدم وجود قيمه للمتغير (x) . كذلك لما كانت هناك أكثر من خطوه مشموله بالتكرار والتي هي الطباعه والقراءه عليه تم تحديدهما بين (begin and end) .

ملاحظه ://

في كل مره يتم قراءة قيمه جديده للمتغير (x) فأن القيمه السابقه ستزول وتحل محلها القيمه الجديده وهذه قاعده عامه يجب أن تلاحظ .

ملاحظه ://

يتم أختيار الشرط بعد الأمر (While) بحيث يساعد حلقة التكرار أن تستمر طالما كان هذا الشرط متحقق , وأن تتوقف الحلقه عن التكرار عندما لا يتحقق هذا الشرط .
في حالة الأمر (Repeat) فأن الشرط ياتي بعد (Until) لذا يجب أن يتم أختياره بحيث عندما يتم فحصه تكون النتيجه (False) أي غير متحقق , لكي يستمر التكرار بالعمل ومتى ما أصبحت نتيجة فحص الشرط (True) فأن التكرار يتوقف .

ملاحظه ://

من السهل كتابة حلقه بشكل عفوي , شرطها لا يصبح متحققا أبدا , هذا سيؤدي الى برنامج مقفل أو مغلق أي يستمر بالتنفيذ الى مالانهايه .

3.6 أيعاز التكرار FOR LOOP

أن هذا الأمر يقوم بتكرار أيعاز أو مجموعة أيعازات لعدد من المرات المحدده مسبقا . والصيغه القواعديه له أذا كان يستخدم لتكرار أيعاز واحد بعده مباشرة هي :

For *{variable}** := *{original value}* **to**/**downto** *{final value}* **do**

{code...(for one action)}

* بشكل عام هذا المتغير يسمى (loop counter)

قيمة هذا العداد تبدأ بالقيمه الأصليه أو الأبتدائيه (original value) ثم تزداد بمقدار واحد (أو تقل بمقدار واحد) في كل مره يتم تنفيذ الأيعاز الذي بعده مباشرة ويستمر كذلك لحين أن تصل قيمة العداد الى القيمه النهائيه (final value) والتي تمثل التنفيذ الأخير .

أما (to / downto) فتستخدم واحده منهما فقط فاذا كانت القيمه الأبتدائيه أقل من القيمه النهائيه فعند ذلك نستخدم (to) وتتغير قيمة المتغير بأضافه واحد له في كل تنفيذ , أما أذا كانت القيمه الأبتدائيه أكبر من القيمه النهائيه فنستخدم (downto) وتتغير قيمة المتغير بطرح واحد منها في كل تنفيذ .

ملاحظه ://

كما في (If , Else , While) فان الأمر (For) لاينفذ أكثر من أيعاز أو عباره تاتي بعده مباشره , فأذا كان هناك أكثر من أيعاز يجب أن يكرر ضمن الأمر (For) فيجب أن يحدد بين (begin and end)

```
Program CH3_Program4;

Begin
  Writeln ('for loop');   {somewhat boring writing all this!!!}
  Writeln ('for loop');       هذا يـكـون ممل لـكـتـابـه كل ذلك
  Writeln ('for loop');
  Writeln ('for loop');
  Writeln ('for loop');
  Writeln ('for loop');
  Writeln ('for loop');
  Readln;
End.
```

ممكن أعادة كتابه هذا البرنامج بطريقه أسهل و أسرع ويؤدي نفس الغرض كما يلي :

```
Program CH2_Program5;

Var Counter: Integer;       {loop counter declared as integer}

Begin
  For Counter: = 1 to 7 do   {it's easy and fast!}
    writeln ('for loop');
  Readln;
End.
```

ملاحظه ://:

لاتستخدم الفارزه المنقوطه بعد الأمر (for ... do) , الأمر (while .. do) , والأمر (repeat) .

3.7 أستخدام (FOR) المتداخله NESTED FOR

ممكن أستخدام الأمر (For) بشكل متداخل ولأكثر من مره وبهذه الحاله فان حلقة (For) تكرر كاملة بعدد مرات التكرار المحدده في (For) الخارجي . فمثلا لو كان لدينا عدد من الطلاب في صف معين (30 طالب مثلا) ونرغب أن نطبع أسماء الطلبه مع الدرجات التي حصل عليها كل منهم في كل الدروس التي يدرسوها في تلك المرحله (8 دروس مثلا) . هنا يجب طباعة أسماء الطلبه وهي 30 أي أن أمر الطباعه سيكرر 30 مره لذا نستخدم (for) لهذا الغرض لأن عدد مرات التكرار محدد , وفي كل مره(أي لكل طالب) يجب أن نطبع الدرجات (8 درجات) أي أن أمر طباعة الدرجات يكرر 8 مرات عليه نستخدم (For) أيضا لطباعة الدرجات لكل طالب , وسيكون البرنامج كما يلي :

```
Program CH3_Program6;
Var
  Name: string;
  Degree, i, j: integer;
Begin
  For i: = 1 to 30 do
    Begin
      Writeln ('enter student name and his/her degree') ;
      Readln (name);
      Writeln (name);
      For j: = 1 to 8 do
        Begin
          Writeln('Enter degree:',j);
          Readln (degree ) ;
          Write (degree:5 ) ;
        End;  {second for }
    End;      {first for }
End.
```

شرح البرنامج ://

في البرنامج أعلاه فأن (for) الأولى تستخدم لطباعة أسماء الطلبه , ولما كان كل طالب له 8 درجات فأن أمر تكرار لهذه الدرجات سيكون من ضمن (for) الأولى (أي عند طباعة أسم طالب معين يجب أن نطبع معه درجاته الثماني قبل الأنتقال الى الطالب التالي) . وبما أن عدد الخطوات المشموله بالتكرار ضمن (for) الأولى هي أكثر من واحده لذا تم تحديدها بين (begin , end) ونفس الشيء بالنسبه للأمر (for) الثانيه . وفي كل مره ننفذ (for) الأولى سننفذ (for) الثانيه كاملة قبل أن ننتقل الى زيادة العداد (I) (أي أن العداد (j) يبدأ بقيمة البدايه ويستمر بالعمل حتى ينتهي بقيمة النهايه في كل زياده للعداد (I)). هذا مشابه لعقارب الساعه فلكي يتحرك عقرب الساعات خطوه واحده فأن عقرب الدقائق يجب أن يتحرك 60 خطوه , وكأنما عقرب الساعات هو (for I : = 1 to 60 do) وهو حلقة تكرار خارجي وعقرب الدقائق هو حلقة التكرار الداخلي (for j : = 1 to 60 do) .

ملاحظه ://

يستخدم الأمر (break) والأمر (continue) مع حلقات (for) وكافة حلقات التكرار الأخرى مثل (repeat , while) وكما يلي :

1. **الأمر (break) ويستخدم للسيطره على تدفق تكرار العبارات وهي تؤدي الى أنهاء أو توقف التكرار , مثال**

```
For i: = 1 to 10 do
 begin
    Readln ( x );
     If  x < 0 then
        Break
     Else
        Writeln ( sqrt ( x ) );
 End;
```

في هذه الحاله يتوقف التنفيذ عند ورود عدد سالب لعدم أمكانية أيجاد الجذر التربيعي للعدد السالب .

2. **الأمر (continue) ويستخدم أيضا مع حلقات التكراروهو يعني أستمر مع حلقة تكرار جديده(أي أهمل تنفيذ الأوامر التي بعد الأمر (continue) عند تحقق شرط معين حيث سيعيد المؤشر الى الأمر (for)) , مثال**

```
For  i: = 1 to 10 do
 Begin
     Readln ( x );
      If  x < 0 then
          Continue
      Else
          Writeln  ( sqrt ( x ) );
  End;
```

في هذه الحاله عند ورود عدد سالب فأن الأمر (continue) سيمنع متابعة تنفيذ العبارات الأخرى في حلقة التكرار والمتمثله بأمر الطباعه في هذا المثال ويعيد المؤشر الى الأمر (for) ليبدأ بتكرار جديد .

ملاحظه ://:

من الممكن أن يكون التداخل بين عبارات التكرار جميعا سواء المتشابهات أو المختلفات , مثلا بين (for , and for) , (for , and while) , (for . and repeat) , (while , and repeat) , (while , and while) , (repeat , and repeat)

3.8 عبارة أختيار الحاله The Simple Case statements

في بعض الاحيان تستخدم (If) المتداخله ولمرات عديده بشكل ممكن أن يكون مطولا أو مملا , ولتسهيل العمل فأنه يمكن أن نستعيض عنها بعبارة (Case) والشكل القواعدي لها هو :

```
Case {variable of type: integer or character ONLY} of

 {input statement- within inverted commas if of type char} :

              {code..}

 {input statement- within inverted commas if of type char} :

              {code..}

   ...

End;  {End Case}
```

ملاحظه ://:

الأمر (Case) ينتهي دائما بالأمر (end) بالرغم من عدم وجود (begin)

ملاحظه ://:

يأتي بعد الأمر (Case) متغير وهذا المتغير من نوع <u>الأعداد الصحيحه أو الحروف</u> فقط ولا يمكن أن نستخدم السلاسل الحرفيه هنا .

ملاحظه ://

يفضل أستخدام الأمر (case) في البرامج التي تحتاج الى ثلاث عبارات (If) متتاليه أو أكثر .

لنرى الفرق بين أستخدام (If) و (Case) من خلال البرنامج التالي والذي يحاكي أستخدام الحاسبه الجيبية ذات العمليات الأربع (Calculator) :

```
Program CH3_Program7;
Var
  ch: char  ;
  num1, num2: integer;
  Result: real;
Begin
  Writeln (' enter two numbers ');
  Readln (num1, num2);
  Writeln (' enter one of operators  " +, - , * , /  "  ');
  Readln (ch);
  If (ch =  ' + ') then
    Result: = num1 + num2
  Else
    If (ch = ' - ') then
      Result: = num1 - num2
    Else
      If (ch = ' * ') then
        Result: = num1 * num2
      Else
        Result: = num1 / num2   ;
  Writeln (result);
End.
```

البرنامج أعلاه برنامج بسيط حيث يتم أدخال عددين وأدخال العمليه الرياضيه المطلوب أجراؤها عليها ثم يقوم المترجم بفحص العمليه التي تم أدخالها لينفذ ما مطلوب فيها على الأعداد , وأخيرا تطبع النتيجه.

الان نعيد كتابة البرنامج أعلاه ولكن بأستخدام (Case of) ونلاحظ الفرق , مع العلم أن النتائج لا تتغير.

```
Program CH3_Program8;
Var
  num1, num2: integer;
  ch: char ;
  Result: real;
Begin
  Writeln (' enter two numbers ');
  Readln (num1, num2);
  Writeln (' enter one of operators " +, - , * ,/ " ');
  Readln (ch);
  Case ch of
    '+ ': result: = num1 + num2;
    '- ': result: = num1 - num2;
    '* ': result: = num1 * num2;
    '/ ': result: = num1 / num2;
  End;
  Writeln (result);
End.
```

ملاحظه ://

لا تستخدم (IF) بعد (Else) عندما يكون هناك أحتمال واحد متبقي , وتستخدم بعد (Else) أذا كان هناك أكثر من أحتمال واحد ويجب الأختيار منهما .. لأن أستخدامها مع وجود أحتمال واحد يعتبر غير منطقي بالرغم من أن البرنامج ممكن أن ينجز .

شرح البرنامج ://

البرنامج أعلاه (8) أكثر بساطه من البرنامج السابق (7) .
لنلاحظ كيفية أستخدام الأمر (case) حيث بعد أن يتم أعطاء قيمه للمتغير (ch) , يتم فحص هذه القيمه بواسطة (case) وكأن العباره تترجم (أذا كانت قيمة ch مايلي أعمل الخطوات التي تقابله) , ثم نكتب القيم التي ممكن أن تكون عليها (ch) حسب متطلبات البرنامج , كل قيمه في سطر منفرد وتوضع بعدها النقطتين المتعامدتين (:) (colon) بعد ذلك نكتب الاجراء الذي يجب أن يحصل عند تحقق أدخال هذا المتغير . فمثلا أذا كانت قيمة المتغير (ch) هي (*) فأن المترجم يفحص أولا (+) وسوف يجدها لا تساوي قيمة المتغير (ch) أي أن الأجابه هي خطأ (false) فيتركها ليقارن القيمه اللاحقه وهي (-) وأيضا سيجد أن الأجابه (false) فيستمر بفحص القيمه التي بعدها وهي (*) هنا ستكون النتيجه (true) , لذا سينفذ العباره أو العبارات التي بعدها وهي أجراء عمليه الضرب ووضع النتيجه بالمتغير (result) . أما الأمر (end) فينهي كافة عمليات الفحص , ليأتي بعدها أمر طباعة النتيجه.

ملاحظه ://

دائما الحروف والسلاسل الحرفيه عند أستخدامها وكتابتها في البرامج على أساس أنها حروف أو سلاسل حرفيه وليس لغرض أخر فأنها تحدد بين علامتي أقتباس .

ملاحظه ://

كما في (for, if ,while , else) والتي تنفذ عباره واحده بعدها فقط , كذلك في حالة (case) فأنها ستنفذ عباره واحده من العبارات التي تأتي بعد عبارات الأدخال (والتي تمثل قيم المتغير) , ولذا أذا كانت هناك أكثر من عباره يجب أن تنفذ بعد قيمة المتغير التي تتطابق مع القيمه المدخله فيجب أن تحدد بين (begin , end) .

ملاحظه ://

بالأمكان أستخدام (else) مع (case) كخطوه أخيره بعد أنتهاء عمليات فحص القيم الأفتراضيه للمتغير , وهي تعني تنفيذ الأجراء الذي يأتي بعد (else) في حالة الفشل في مطابقة القيمه المدخله للمتغير مع القيم المفروضه .

```
Program CH3_Program9 ;
Var
  num1, num2 : integer ;
  Ch: char;
  Result: real;
Begin
  Writeln (' enter two numbers ') ;
  Readln (num1, num2);
  Writeln (' enter one of operators " +, - , *, /, "  ');
  Readln (ch);
  Case ch of
    '+ ': result: = num1 + num2;
    '- ': result: = num1 - num2;
    '* ': result: = num1 * num2;
    '/ ': result: = num1 / num2;
    Else
      Writeln (' Error in entering operator ');
  End;
  Writeln (result);
End.
```

3.9 جملة IN

تستخدم جملة (in) في الجمل الشرطيه أستخدامات مختلفه وهي :

أولا :// لتحديد المدى الذي يعمل به متغير معين ولتوضيح ذلك لنرى المثال التالي :

```
If ( degree > = 90 ) And ( degree <= 100 ) then
  Writeln ( ' Excellent ' ) ;
```

نلاحظ أن الشرط في المثال يحدد الدرجه بين (90 لغاية 100) ليعطي درجة الأمتياز, من الممكن أعادة كتابه الشرط أعلاه بأستخدام أيعاز تحديد المدى (in) , وكما يلي :

```
If degree in [ 90 .. 100 ] then
  Writeln ( ' Excellent ' ) ;
```

لكي نستخدم الأيعاز أو الأمر (in) لتحديد المدى نستخدم بعده قوسين مربعين وتحدد بداية ونهاية المدى على أن يفصل بينهما نقطتان كما موضح في المثال أعلاه . أن الترجمه الحرفيه لهذه العباره هي (أذا الدرجه في المدى المحدد بين القوسين المربعين فأكتب كذا) .

ثانيا :// ممكن أن نستخدمها بطريقه أخرى , فبدلا من أن نحدد المدى كبدايه ونهايه يمكن الأستعاضه عنها بقيم معينه بحيث أن المتغير ممكن أن يكون واحد من هذه القيم **. مثال**

```
If x in [ 5 , 10, 20, 35, 50 ] then
  Writeln ( ' OK ' ) ;
```

هنا هذا يعني أذا كانت قيمة (x) هي أحدى القيم المحدده بين القوسين المربعين فأكتب (OK) أي أن عملية التطابق ستكون مع واحده من هذه القيم بينما في الحاله الأولى فأن أي قيمه بين القيمتين المحددتين تحقق الشرط .

ثالثا :// الحاله الثالثه لاتختلف عن الحالتين السابقتين ولكن بدل من أستخدام الأرقام ممكن أستخدام الحروف . مثال

```
If ch in [ 'a' .. 'z' ] then
  Writeln ( ' small character ' ) ;

OR

If ch in [ 'a' , ' b ' , 'c ' ] then
  Writeln ( ' OK ' ) ;
```

ملاحظه ://

ممكن أن نستخدم النقطتين المتجاورتين (..) للدلاله على المدى بدون الأمر (in) كما نعمل مع (case)
مثل :

```
Case  x of
 90 .. 100 : writeln ( ' Excellent ' ) ;
```

3.10 أمثله محلوله

- أكتب برنامج لأيجاد الرقم الأكبر بين رقمين .

```
Program CH3_Program10;
Var
  X,y:integer;
Begin
  Writeln ('Enter two numbers');
  Readln(x, y);
  If(x>y) then
    Writeln ('the largest number =', x)
  Else
    Writeln ('the largest number =, y);
End.
```

- أكتب برنامج لأيجاد قيمة (z) حيث أن

$$Z = 5x^2 + 3x/y \quad \text{when } x >= y$$

$$\text{and} \quad Z = y^2 - 3x \quad \text{when } x < y$$

```
Program CH3_Program11;
Var
  X, y: integer;     z: real;
Begin
  Writeln ('Enter x and y ');
  Readln(x);     readln(y);
  If(x>=y) then
    Z: =5*sqr(x) + 3*x/y
  Else
    Z: =sqr(y)-3*x;
  Writeln (z);
End.
```

- أكتب برنامج لطباعة الأرقام الفرديه المحدده بالرقمين (35 – 55) .

```
Program CH3_Program12;
Var
  I:integer;
Begin
  For i: = 35 to 55 do
    begin
      If (i mod 2<>0) then
        Continue ;
      Write (i: 4);
    End;
End.
```

- أكتب برنامج لأيجاد مجموع الأرقام الزوجيه المحدده بين الرقمين (100 – 2) .

```
Program CH3_Program13;
Var
  I, sum: integer;
Begin
  Sum: =0;
  For i: =2 to 100 do
    If (I mod 2=0) then
      Sum: =sum+I;
  Writeln (sum);
End.
```

- أكتب برنامج لأيجاد أكبر وأصغر عدد من بين (15) عدد .

```
Program CH3_Program14;
Var
  X,max,min:integer;
Begin
  Writeln ('Enter first number');
  Readln(x);
  Max: =x;        min: =x;
  For i: =1 to 14 do
    Begin
      Readln(x);
      If(x>max) then
        Max: =x;
      Else
        If(x<min) then
          Min: =x;
    End;
  Writeln ('max number=', max);
  Writeln ('min number=', min);
End.
```

- أكتب برنامج لأيجاد مجموع عدد من الأرقام أخر رقم فيها يساوي (0) .

```
Program CH3_Program15;
Var
  Sum, x: integer;
Begin
  Sum: =0;
  Repeat
    Writeln ('Enter new number');
    Readln(x);
    Sum: =sum+x;
  Until(x=0);
  Writeln (sum);
End.
```

- أكتب برنامج لأيجاد معدل مجموعه من الأرقام أخر رقم فيها هو(12) .

```
Program CH3_Program16;
Var
  Sum, x, count: integer;
Begin
  Sum: =0;     count: =0;
  Writeln ('Enter first number in group');
  Readln(x);
  While(x<>12) do
    Begin
      Sum: =sum+x;
      Inc (count);
      Readln(x);
    End;
  Writeln (sum/count);
End.
```

- أكتب برنامج لأيجاد ناتج (n) من العناصر في المعادله
2/1*2/3*4/3*4/5*6/5*6/7………

```
Program CH3_Program17;
Var
  I, n: integer;      sum: real;
Begin
  Writeln ('Enter number of elements ');
  Readln (n);
  Sum: =1;
  For i: =1 to n do
    Begin
      If (I mod 2=0) then
        Sum: =sum * i/ (i+1);
      Else
        Sum: =sum * (i+1)/i;
    End;
  Writeln (sum);
End.
```

- أكتب برنامج لأيجاد العدد الأصغر بين ثلاث أعداد

```
Program CH3_Program18;
Var
  X,y,z:integer;
Begin
  Writeln ('Enter three numbers');
  Readln(x, y, z);
  If(x<y) and (x<z) then
    Writeln ('min number=', x)
  Else
    If(y<x) and (y<z) then
      Writeln ('min number=',y)
    Else
      Writeln ('min number=',z);
End.
```

- أكتب برنـامج لقراءة عدد ثم أوجد مجموع أرقامـه والرقم الأكبر بين أرقامـه. (مثلا العدد 5472 فأن مجموع أرقامه هي (18) والرقم الأكبر فيه هو (7))

```
Program CH3_Program19;
Var
  X,z:integer;
Begin
  Writeln ('Enter number');
  Readln(x);
  Repeat
    Z: = x mod 10;
    Write (z);
    If (z > max) then
      Max: =z;
    X: =x div 10;
  Until(x=0);
  Writeln ('max number=', max);
End.
```

- أكتب برنامج لتحويل الرقم العشري (decimal number) الى ثنائي (binary number).

```
Program CH3_Program20;
Var
  Sum, I, x, b: integer;
Bagin
  Sum: =0;        i: =1;
  Writeln ('Enter decimal number');
  Readln(x);
  While(x<>0) do
    Begin
      B: =x mod 2;
      Sum: =sum+ i*b;
      X: =x div 2;
      I: =i*10;
    End;
  Writeln (sum);
End.
```

- أكتب برنامج لطباعة مايلي :

```
0 3 6 9 … n
3 6 9 … n
6 9 … n
9 … n
.
n
```

```
Program CH3_Program19;
Var
  N,x,x1:integer;
Begin
  Writeln ('Enter the last number N');
  Readln (n);
  If (n mod 3 = 0) then
    Begin
      X: =0;
      While(x<=n) do
        Begin
          X1:=x;
          Repeat
            Write(x1:6);
            Inc(x1,3);
          Until(x1>n);
          writeln;
          x:=x+3;
        End;  {while}
    End   { if }
    Else
      writeln('Error, number N should divede by 3');
End.
```

- أكتب برنامج لطباعة الشكل التالي :

```
     *
     *
    *
   *
    *
     *
      *
```

```
Program CH_3 Program20;
Var
  I:integer;
Begin
  For i: = 3 downto -3 do
    Writeln ('*': 10 + abs (i));
End.
```

الفصل الرابع

المصفوفات ARRAYS

4.1 المقدمه

سيتم التركيز في هذا الفصل على المصفوفات بنوعيها الأحادي والثنائي مع أمثله توضيحيه لقراءة وطباعة المصفوفات والعمليات التي ممكن أجراؤها عليها.

4.2 المصفوفات ARRAYS

المصفوفه هي هيكل بيانات يخزن مجموعه من المتغيرات لها نفس النوع . هي مشابهه لمجموعه من الصناديق مرتبطه واحده بالأخرى بشكل متسلسل وكل واحد من هذه الصناديق ممكن أن يحتوي على بيانات معينه وجميع البيانات في هذه الصناديق هي من نوع واحد . ويقال عن المصفوفه أنها هيكل بياني ثابت (static) وذلك لأن المبرمج عندما يعلن عن نوع وحجم المصفوفه في حقل الأعلان عن المتغيرات , فان حجمها سيبقى ثابت في البرنامج ولا يمكن تغييره , وهناك المصفوفات الديناميكيه (dynamic) التي من الممكن أن نغير حجمها أثناء تنفيذ البرنامج . نستخدم للمصفوفه متغيرا مفردا فقط كاداة لخزن البيانات , وهذا المتغير يخزن عنوان الموقع الاول في الذاكره الذي تخزن فيه المصفوفه وباقي المواقع تأتي بعده بالتتابع (متسلسله) .

المصفوفات ://

هي مجموعه مرتبه من البيانات والتي قد تحتوي على عدد ثابت من العناصر أو غير ثابت , وتستخدم أسلوب العنوان المحسوب لأيجاد موقع الخليه المطلوبه في الذاكره وذلك عن طريق معادلات رياضيه.

بشكل عام فأن المصفوفه نوعان (هناك مصفوفات متعدده المستويات هي خارج نطاق هذا الكتاب) , مصفوفه أحاديه ومصفوفه ثنائيه .

4.2.1 المصفوفه الأحاديه ONE DIMENSION ARRAY :

نبدأ هنا مرحله جديده من البرمجه . لغاية الأن لسنا قادرين على معالجه وخزن كميات كبيره من البيانات بطريقه مناسبه . فمثلا أذا أردنا العمل على قائمه طويله من الأعداد أو الأسماء , فأننا سنعلن عن متغيرات منفصله لكل عدد أو أسم . لحسن الحظ فأن لغة البرمجه باسكال ولغات البرمجه الأخرى توفر عدد من المتغيرات المهيكله لتسهيل حل المشاكل التي تحتاج الى العمل مع كميه كبيره من البيانات , ببساطه فأن المتغير من هذا النوع يستخدم معرف واحد لخزن كميه كبيره من البيانات في الذاكره .

المصفوفه مصممه لحمل كميه كبيره من البيانات من نفس النوع بطريقه منظمه . أن أستخدام المصفوفه يسمح بحجز مجموعه من مواقع الذاكره المتتاليه والتي يمكننا معالجتها ككتله واحده أو مكونات منفصله .
أذن المصفوفه الأحاديه هي التي من الممكن كتابة عناصرها على شكل صف أو عمود واحد ويكون عدد العناصر وبالتالي عدد المواقع في الذاكره لخزن هذه المصفوفه فيها مساويا الى حجم المصفوفه .

الأعلان عن المصفوفه ياخذ الشكل التالي :

*<arrayName> : **Array** [n..m] **Of** <Data Type>;*

ممكن في حقل المتغيرات تأخذ الشكل التالي :

Var

*myArray : **Array** [1..20] **of** Integer;*

حيث يتم الأعلان عن المصفوفه في حقل المتغيرات ويكون :

أولا :// بأعطاء أسم للمصفوفه هنا (myarray) وهو متغير ويتم أختياره من قبل المبرمج , ثم تأتي النقطتان المتعامدتان كما في تعريف المتغيرات .

ثانيا :// نكتب كلمة مصفوفه (array) للدلاله على أن هذا المتغير هو من نوع مصفوفه والمصفوفه يجب أن يكون حجمها محدد قبل البدأ بتنفيذ البرنامج لذا سيكون التعريف (مصفوفه حجمها كذا وعناصرها من النوع المحدد) , عملية تحديد الحجم يكون بأستخدام أقواس مربعه وبداخلها الحجم وهو يتكون من جزئين تفصل بينهما نقطتان فقط (للدلاله على المدى الذي تعمل به المصفوفه) , الجزء الأول يمثل رقم البدايه والثاني يمثل رقم النهايه , وعليه الفرق بينهما يمثل عدد العناصر في المصفوفه (حجم المصفوفه) . أن حجم المصفوفه يحدد من قبل المبرمج وفقا لمتطلبات البرنامج.
أن أستخدام المصفوفه مشابه لأستخدام المتغيرات الأعتياديه وبدلا من أن نستخدم عشرين متغير كما في المثال أعلاه من نوع أعداد صحيحه , يمكن أن نستخدم متغير واحد كمصفوفه (أي كمجموعة متغيرات حسب حجم المصفوفه (سيتم حجز مواقع في الذاكره وفقا لذلك) , لها نفس الأسم (أي أن جميع هذه المواقع لها ذات الأسم) وتختلف بالفهرسه (index) أي لكل واحد منها رقم يشار به للموقع (للتمييز بين متغير وأخر)) , أن العمل مع المصفوفات يساعد على تسهيل العمل مع المتغيرات والمحافظه على قيم العديد من المتغيرات وتمكن المبرمج من أجراء أعمال كثيره بيسر وسهوله .

- **كيفية التعامل مع المصفوفه الأحاديه :**

لنأخذ المثال أعلاه والذي أعلنا فيه عن عشرين عنصر فأننا يمكننا الوصول الى كل عنصر فيها وكما يلي :

أولا : // **أسناد قيم لبعض عناصر المصفوفه :**

حيث يمكن أسناد قيمه من نوع الأعداد الصحيحه (حسب تعريف المصفوفه) لأي عنصر في المصفوفه وذلك من خلال تحديد رقم أو تسلسل العنصر في المصفوفه ثم المساواة مع القيمه التي من المفترض وضعها فيه (أو من خلال أستخدام أيعاز القراءه) وكما يلي :

<arrayName> [index] := <relevant data>

```
myArray[5] := 10;
myArray[1] := 25;
OR
Readln ( myArray[1] ) ;
```

هنا تم أسناد القيمه (10) الى المتغير أو الموقع الخامس (لاحظ كيفية الأشاره الى موقع العنصر , يوضع رقم الموقع بين قوسين مربعين) بينما تم أسناد القيمه (25) الى الموقع الأول وستبقى هذه القيم لهذه المواقع ألا أذا تم تغييرها في البرنامج , فعندما نريد طباعة محتويات الموقع الأول فسنجد فيه القيمه (25). أن طريقة أدخال بيانات الى عناصر المصفوفه هي ذات الطرق التي سبق وأن بيناها لأدخال بيانات الى المتغيرات الأعتياديه . لنلاحظ المثال التالي :

```
Program CH4_Program1 ;
Var
  myVar   : Integer;
  myArray : Array[1..5] of Integer;
Begin
  myArray[2] := 25;
  myVar := myArray[2];
End.
```

هنا المتغير (myvar) هو متغير أعتيادي بينما المتغير (myarray) هو مصفوفه حجمها خمسة عناصر, أثناء تنفيذ البرنامج تم أدخال القيمه (25) للموقع الثاني في المصفوفه , وفي الخطوه اللاحقه تم مساواة العنصر الثاني من المصفوفه مع المتغير (myvar) وهذا سيؤدي :

- تعويض قيمة محتويات الموقع الثاني في المصفوفه بدلا عن المتغير (myarray[2]) أي سيكون الطرف الأيمن من المعادله يحتوي على العدد الصحيح (25) .
- ستؤول هذه القيمه الى الطرف الأيسر من المعادله (أي أن المتغير myvar تكون قيمته هي 25) . وهي نفس الطريقه التي نتعامل بها مع المتغيرات الأعتياديه.

ثانيا :// أسناد قيم لكافة عناصر المصفوفه (قراءة المصفوفه)

في كل البرامج التي تحتوي على مصفوفات يجب القيام بأدخال قيم لعناصر المصفوفه قبل أن نتعامل معها , وأن عملية أدخال قيم لعناصر المصفوفه تتم أما من داخل البرنامج أو من خارجه كما كان الحال مع المتغيرات الأعتياديه . لنلاحظ المثال التالي والذي يحتوي على مصفوفتين منفصلتين دون وجود أي علاقه بينهما حيث أن الأولى من نوع الأعداد الصحيحه والثانيه من نوع

العبارات المنطقيه وسنضع القيمه (0) لكل عناصر المصفوفه الاولى ونضع القيمه (false) لكل عناصر المصفوفه الثانيه كما في ادناه :

```
Program CH4_Program2 ;
Var
  i            : Integer;
  myIntArray   : Array[1..20] of Integer;
  myBoolArray  : Array[1..20] of Boolean;
Begin
  For i := 1 to 20 do
    Begin
      myIntArray[i]  := 0;
      myBoolArray[i] := false;
    End;
End.
```

ثالثا :// طباعة المصفوفه

نحتاج في الكثير من البرامج الى طباعة المصفوفه , وطبعا يجب أن تكون المصفوفه غير خاليه (أي أن عناصرها أو مواقعها تحتوي على قيم) .
أن عملية كتابة أو طباعة عناصر مصفوفه معينه لا تختلف عن قراءة المصفوفه عدا أن الأيعاز الخاص بالقراءه يستبدل بالأيعاز الخاص بالطباعه أو الكتابه (أي أن (readln OR :=)) يتم أستبدالها بالأيعاز أو الأمر (write OR writeln) .

```
Program CH4_Program3;
Var
  i           : Integer;
  myIntArray: Array[1..20] of Integer;
  myBoolArray: Array[1..20] of Boolean;
Begin
  For i: = 1 to 20 do
    Begin
      Writeln (myIntArray [i]);
    End;
End.
```

ملاحظه :

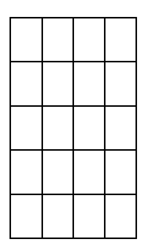
العمليات الرياضيه التي تجرى على عناصر المصفوفه هي ذات العمليات الرياضيه التي تجرى على ذلك النوع (نوع البيانات لعناصر المصفوفه المعلن عنه في حقل الأعلان عن المتغيرات) .

- **المصفوفه المضغوطه Packed Array**

المصفوفه مفيده جدا لحمل كميه كبيؤه من البيانـات , واحده من سلبيات أستخدام المصفوفه هـي أحتياجهـا الـى كميـه كبيـره مـن الـذاكره , عمليـا المصفوفه مـن نـوع الأحـرف (التـي عناصـرها حروف) تحجز ذاكره أكبر من الأنواع الأخرى .. لشرح ذلك لنرى التعريف التالي :

```
Var
   X : array [ 1..5 ] of char ;
```

بالتأكيد فأن هذا التعريف سيحجز عدد من المواقع في الذاكره بما يتناسب وحجم المصفوفه , هنا سيتم حجز خمسة مواقع (موقع لكل عنصر) ولما كان نوع العناصر من النوع الحرفي فسيعامل كل حرف ككلمه (word) في الذاكره , وكل كلمه تتكون من عدد من البايتات . لو فرضنا أن الكلمه تتكون من أربع بايتات فأن المصفوفه ستحجز (20) بايت كما يلي :



فأذا أسندنا كلمة (HELLO) كعناصر للمصفوفه فستكون المصفوفه بالشكل التالي :

H			
E			
L			
L			
O			

نلاحظ أن هناك خمسة عشر بايت غير مستغله , وهذا سوء أستخدام للذاكره , لذا فأن لغة البرمجه باسكال أوجدت الأيعاز (Packed Array) لتحدد المساحه الخزنيه بأقل حجم ممكن وكما يلي :

```
Var
   X : packed array [ 1..5 ] of char ;
```

هذا التعريف سيولد المصفوفه التاليه بعد أسناد الكلمه (HELLO) الى عناصر المصفوفه:

H	E	L	L	O			

هنا تم أستخدام كلمتين فقط (أي 8 بايت) .

4.2.2 **المصفوفه الثنائيه 2-D ARRAY :**

المصفوفه الثنائيه هي المصفوفه التي تكتب عناصرها على شكل صفوف وأعمده في ذات الوقت (أي يكون هناك عدد من الصفوف وكل صف فيه عدد من العناصر وكذلك عدد من الأعمده وكل عمود فيه عدد من العناصر) , أن عدد العناصر في المصفوفه الثنائيه وبالتالي عدد المواقع في الذاكره يساوي حاصل ضرب عدد الصفوف في عدد الأعمده . يتم الأعلان عن المصفوفه الثنائيه بنفس الطريقه التي يتم فيها الأعلان عن المصفوفه الأحاديه مع فارق واحد فقط وهو أن الأعلان عن عدد العناصر في المصفوفه الثنائيه يتم من خلال قيمتين داخل القوسين المربعين تفصل بينهما فارزه كل واحده لها المدى الخاص بها وليس قيمه واحده كما في المصفوفات الأحاديه ويحتوي المدى الأول على عدد الصفوف تبدأ من الرقم واحد وتنتهي بالرقم الذي يمثل العدد الكلي للصفوف (وهي تمثل عدد العناصر في العمود الواحد) , أما المدى الثاني فيحتوي على عدد الأعمده وبنفس الطريقه المستخدمه لعدد الصفوف (وهي تمثل عدد العناصر في الصف الواحد). أما الأشاره الى عنصر معين في المصفوفه فيتم من خلال أسم المصفوفه وقوس مربع يحتوي قيمتين الأولى تمثل رقم الصف والثانيه تمثل رقم العمود (***ولا يجوز أن تكون القيمه الأولى للأعمده والثانيه للصفوف***).

طريقة الأعلان عن المصفوفات الثنائيه كما يلي :

my2DArray : **Array** [1..i , 1..j] **of** <*DataType*>;

هي ذات الطريقه المستخدمه في الأعلان عن المصفوفات الأحاديه عدا الفرق الذي سبق أن أشرنا اليه .

تمثيل المصفوفه الثنائيه (5 x 5) منطقيا يكون على شكل شبكه كما يلي :

1	2	3	4	5
2				
3			3,4	
4				
5		5,3		5,5

حيث أن أرقام المربعات في الصف الأفقي ذات اللون الأحمر تمثل أرقام الأعمده , بينما أرقام المربعات في الصف العمودي ذات اللون الأحمر تمثل أرقام الصفوف . أما الأرقام الموجوده في المربعات ذات اللون الأزرق فهي تمثل رقم الصف والعمود لذلك الموقع (الرقم الأول هو رقم الصف والثاني هو رقم العمود) , (فمثلا الرقم (3 , 4) يمثل الموقع أو العنصر الموجود في الصف الثالث والعمود الرابع من المصفوفه الثنائيه) .

- **كيفية التعامل مع المصفوفه الثنائيه :**

أن التعامل مع المصفوفات الثنائيه مشابه لطريقة التعامل مع المصفوفات الأحاديه مع الأخذ بنظر الأعتبار خصوصية المصفوفه الثنائيه .

أولا ://: أسناد قيم لبعض عناصر المصفوفه

ويتم ذلك أما بأستخدام المساواة أو أستخدام أيعاز القراءه , مثال

My2DArray [3, 4] := 44;
My2DArray [5, 3] := 20;

OR
Readln (My2DArray [1, 2]);

حيث سيتم وضع القيمه (44) في الموقع الذي يتقاطع فيه الصف الثالث مع العمود الرابع , وتوضع القيمه (20) في الموقع الذي يتقاطع فيه الصف الخامس مع العمود الثالث (لاحظ الشكل أعلاه) .   أما كيفية الأعلان عنها برمجيا يكون :

```
Program CH4_Program4;
Var
  my2DArray: Array [1..3 , 1..5] of Byte;
Begin
  my2DArray [2, 4]:= 10;
  myVar := my2DArray [3,4];
End.
```

المصفوفه أعلاه في المثال 4 من نوع (byte) أي أن كل عنصر سيمثل ببايت واحد في الذاكره . وسنمثلها كما يلي :

1	2	3	4	5
2			10	
3				

ثانيا ://: أدخال قيم لكافة عناصر المصفوفه (قراءة المصفوفه)

أن أدخال قيم لعناصر المصفوفه الثنائيه يحتاج برمجيا الى أستخدام أوامر التكرار (For) مرتين وبشكل متداخل , حلقة التكرار الأولى للصفوف والثانيه للأعمده . كما يلي :

```
Program CH4_Program5;
Var
  my2DArray: Array [1..3 , 1..5] of integer;
  I, j: integer;
Begin
  For I: = 1 to 3 do
  For j: = 1 to 5 do
    Readln (my2DArray [i, j]);
End.
```

في هذا البرنامج أستخدمنا أمر الأدخال (readln) لأنجاز عملية الأدخال بدلا من المساواة والتي أستخدمت في البرنامج (2) لنتعلم الطريقتين .

ثالثا ://: طباعة المصفوفه الثنائيه

تتم طباعة المصفوفه الثنائيه بنفس طريقة قراءة المصفوفه الثنائيه مع ملاحظة أستبدال أمر القراءه بأمر الطباعه , لاحظ البرنامج التالي :

```
Program CH4_Program6;
Var
  my2DArray: Array [1..3 , 1..5] of integer;
  I, j: integer;
Begin
  For I: = 1 to 3 do
  For j: = 1 to 5 do
    writeln (my2DArray [i,j] ) ;
End.
```

ستكون مخرجات هذا البرنامج هي طباعة كل قيمه على سطر منفصل (أما أذا أستخدمنا الأمر write فأنه سيطبع كل القيم على ذات السطر) , أذن كيف يمكننا طباعة المصفوفه الثنائيه على شكل شبكه , لاحظ البرنامج التالي :

```
Program CH4_Program7;
Var
  my2DArray: Array [1..3 , 1..5] of integer;
  I, j: integer;
Begin
  For I: = 1 to 3 do
    Begin
      For j: = 1 to 5 do
        Write (my2DArray [i, j],'  ');
      Writeln;
    End;
End.
```

شرح البرنامج ://

هذا البرنامج لايختلف كثيرا عن البرنامج (6) عدا أضافة أمر طباعه خالى . عند البدأ بالتنفيذ فان التكرار الأول المتمثل بالأمر (for) سينفذ وتكون قيمة (I) تساوي واحد وبما أن حلقة التكرار تبدأ وتنتهي بالأوامر (begin , end) هذا يعني أن كل الكتله المحصوره بين (begin , end) ستنفذ كامله مع كل قيمه للمتغير (I) . أن أول أيعاز يأتي هو حلقة التكرار الثانيه ولما لم يكن بعدها الأمر (begin) فأنها ستنفذ خطوه واحده بعدها فقط وهي أمر الطباعه , هذا الأمر سيطبع محتويات المصفوفه المحدده بين القوسين وحسب الموقع المبين أزائها حيث أن المترجم سيقوم بالتعويض عن قيمتي (I, j) وهما (1 , 1) , ثم يأتي بقيمة هذا الموقع لطباعتها , وبعد طباعة محتويات هذا الموقع يطبع الفراغ الوارد بأمر الطباعه وحسب ما محدد بالأمر(فراغين) .. يرجع المترجم الى التكرار الثاني ليزيد قيمة المتغير (j) بمقدار واحد فتصبح قيمته أثنين لينفذ أمر الطباعه ثانية ويطبع محتويات الموقع (1 , 2) بجانب القيمه الأولى (التي هي رقم وفراغ) لأن أمر الطباعه هو (write) ثم العوده لزيادة قيمة المتغير (j) لتكون مساويه الى (3) وتتم طباعة محتويات الموقع (1 , 3) والفرغ الذي بعده , تزاد قيمة المتغير (j) وتصبح قيمته (4) وتستمر العمليه لحين أن تتم طباعة خمسة قيم جميعها على سطر واحد يفصل بين واحده وأخرى فراغ حسب ما مبين بأمر الطباعه عند ذاك فأن قيمة (j) ستكون مساويه الى (6) لذلك يتوقف تنفيذ الأوامر التابعة لحلقة التكرار الثانيه وينتقل التنفيذ الى الأمر التالي وهو طباعه ... ولما كان أمر الطباعه لايحتوي على قيمه ليطبعها فأنه سيكتفي بالتأشير على سطر جديد دون طباعة أي شيء.. , يعود المؤشر الى حلقة التكرار الأولى لأنها لم تنتهي بعد ويزيد قيمة المتغير (I) ليكون (2) وتبدأ عملية تنفيذ جديده مشابهه لما شرح سابقا لكن مع قيمة المتغير (I) المساويه لأثنين وعلى سطر جديد . وتستمر العمليه لحين الأنتهاء من كامل حلقة التكرار الأولى لينتقل بعدها الى نهاية البرنامج .

4.3 أمثله محلوله

- أكتب برنامج لقراءة مصفوفه أحاديه مكونه من (20) عنصر, ثم أطبعها بشكل معكوس .

```
Program CH4_Program8;
Var
  A: array [1..20] of integer;
  I: integer;
Begin
  For i: =1 to 20 do
    Readln (a[i]);
  For i: =20 downto 1 do
    Write (a[i]:7);
End.
```

- أكتب برنامج لقراءة مصفوفه أحاديه مكونه من (20) عنصر , ثم رتب عناصرها تصاعديا حسب قيمها .

```
Program CH4:Program9;
Var
  A: array [1..20] of integer;
  I, j, temp: integer;
Begin
  For i: = 1 to 20 do
    Readln (a[i]);
  For i: =1 to 19 do
  For j: = i+1 to 20 do
    If (a[i] > a[j]) then
      Begin
        Temp: =a[i];
        A[i]:=a[j];
        A[j]:=temp;
      End;
  For i: =1 to 20 do
    Write (a[i]:6);
End.
```

- أكتب برنامج لقراءة مصفوفتين أحاديتين كل منهما مكونه من (15) عنصر, ثم أدمج المصفوفتين بمصفوفه واحده حجمها (30) عنصر .

```
Program CH4_Program10;
Var
  A, b: array [1..15] of char;
  C: array [1..30]of char;
  I: integer;
Begin
  For i: = 1 to 15 do
    Begin
      Readln (a[i]);
      Readln (b[i]);
    End;
  For i: = 1 to 30 do
    Begin
      If (i<=15) then
        C[i]:=a[i]
      Else
        C[i]:=b [i-15];
    End;
  For i: =1 to 30 do
    Write(c[i]:3);
End.
```

- أكتب برنامج لقراءة مصفوفه ثنائيه حجمها (5x4), ثم أوجد مجموع عناصرها .

```
Program CH4_Program11;
Var
  A: array [1..5, 1..4] of integer;
  I, j, sum: integer;
Begin
  Sum: =0;
  For i: =1 to 5 do
  For j: =1 to 4 do
    Readln (a [I, j]);
  For i: =1 to 5 do
  For j: =1 to 4 do
    Sum: =sum + a [I,j];
  Writeln (sum);
End.
```

- أكتب برنامج لقراءة مصفوفه ثنائيه حجمها (5x4) , ثم أوجد العدد الأكبر من بين عناصر المصفوفه .

```
Program CH4_program12;
Var
  B: array [1..5, 1..4]of integer;
  I, j, max: integer;
Begin
  For i: =1 to 5 do
  For j: =1 to 4 do
    Readln (b [I, j]);
  Max: =b [1, 1];
  For i: =1 to 5 do
  For j: =1 to 4 do
    If (b [I, j] > max) then
      Max: =b [I, j];
  Wrireln (max);
End.
```

- أكتب برنامج لقراءة مصفوفتين كل منهما بحجم (3x4) , ثم أوجد مجموعهما والفرق بينهما .

```
Program CH4_Program13;
Var
  A, b, c, d: array [1..3, 1..4] of integer;
  I, j: integer;
Begin
  For i: =1 to 3 do
  For j: =1 to 4 do
    Begin
      Readln (a [I, j]);
      Readln (b [I, j]);
    End;
  For i: =1 to 3 do
  For j: =1 to 4 do
    Begin
      C [I, j]:=a [I, j] + b [I, j];
      D [I, j]:=a [I, j] - b [I, j];
    End;
  For i: =1 to 3 do
    Begin
      For j: =1 to 4 do
        Write(c [I, j]:6);
      Writeln;
    End;
  For i: =1 to 3 do
    Begin
      For j: =1 to 4 do
        Write (d [I, j]:6);
      Writeln;
    End;
End.
```

- أكتب برنـامج لقراءة مصفوفه حجمها (6x6) , ثم أوجد مجموع عناصر القطر الرئيس للمصفوفه .

```
Program CH4_Program14;
Var
  Cd: array [1..6, 1..6] of integer;
  I, j, sum: integer;
Begin
  Sum: =0;
  For i: =1 to 6 do
  For j: =1 to 6 do
    Readln (cd [I, j]);
  For i: =1 to 6 do
    Sum: =sum + cd [i, i];
  Writeln (sum);
End.
```

- أكتب برنـامج لقراءة مصفوفتين حجم الأولـى (3x4) وحجم الثانيـه (4x5) , ثم أوجد حاصل ضربهما .

```
Program CH4_Program15;
Var
  A: array [1..3, 1..4] of integer;
  B: array [1..4, 1..5] of integer;
  C: array [1..3, 1..5] of integer;
  I, j,k: integer;
Begin
  For i: =1 to 3 do
  For j: =1 to 4 do
    Readln (a [I, j]);
  For i: =1 to 4 do
  For j: =1 to 5 do
    Readln (b [I, j]);
  For k: =1 to 3 do
  For i: =1 to 5 do
    Begin
      C [k, i]:=0;
      For j: =1 to 4 do
        C [k, i]:=c [k, i] + a [k, j]* b[j,i];
    End;
  For i: =1 to 3 do
    Begin
      For j: =1 to 5 do
        Write(c [I, j]);
      Writeln;
    End;
End.
```

الفصل الخامس

الأجراءات والدوال PROCEDURES AND FUNCTIONS

5.1 المقدمه

الأجراءات هي برامج صغيره تستخدم ضمن البرنامج الرئيس , هذا الفصل سيركز على كيفية الأستفاده من الأجراءات والدوال وماهية الضوابط التي تتحكم بهما , مع عدد من الأمثله التوضيحيه .

5.2 الأجراءات PROCEDURES

الأجراءات أو الروتينات الفرعيه كما تسمى أحيانا هي جزء من برنامج توفر فعل خاص . و تستخدم لمساعدة المبرمج لتجنب التكرار في البرامج وخصوصا البرامج الكبيره بشكل عام فأن البرنامج يجزأ الى أجزاء صغيره , وهذه الأجزاء تجزأ الى أجزاء أصغر وهكذا لحين الوصول الى أجزاء سهلة التنفيذ أو الأنجاز . وكما في البرامج الأعتياديه تبدأ الأجراءات مع الأمر (begin) وتنتهي مع الأمر (; end) وممكن أن يكون لها متغيرات خاصه بها لا تستخدم في البرنامج الرئيسي .

يبدأ الأجراء بكلمه (procedure) متبوعه بأسم الأجراء كما هو الحال في كتابة البرنامج , وممكن أن يتبع بقوسين يحتويان على المتغيرات الداخله والخارجه والتي تدعى (الوسائط) (parameters) (وهي تستخدم للتواصل بين النماذج أو الأجراءات) والتي سنأتي عليها لاحقا وهذا يدعى **رأس الأجراء** . وكل مايلي ذلك فهو مشابه للبرامج الأعتياديه والتي سبق وأن تطرقنا لها .

أن الفرق بين البرنامج الأعتيادي والأجراء هو مايلي :

1. من المهم جدا أن يكون حجم البرنامج صغير . أن كتابة البرنامج على شكل أجراءات أو دوال (سنأتي عليها لاحقا) يساعد الى درجه كبيره الى تقليل حجم البرنامج بشكل عام .

2. أن كتابه البرنامج على شكل أجراءات يساعد المبرمج مستقبلا أو أي شخص أخر من متابعه البرنامج وتصحيح أو تعديل البرنامج أذا أقتضت الحاجه بسهوله ويسر .

3. الأجراءات تساعد المبرمج على التخلص من تكرار مقاطع معينه من البرنامج والتي تسبب أشكالات كثيره للمبرمج , أضافه الى زيادة حجم البرنامج .

4. الأجراءات تعطي المبرمج مرونه أكبر لمتابعة الأخطاء كما أسلفنا, وكذلك تساعد على أستخدام هذه الأجراءات في برامج أخرى عند الحاجه لها دون الحاجه الى أعادة كتابتها .

الصيغه العامه للأجراء هي :

Procedure < PROCEDURE_NAME > ;
OR
Procedure <PROCEDURE_NAME (**Var** *Variable_Name* : *Type*) ;

ملاحظه ://

هناك نوعين من المتغيرات , متغيرات تستخدم داخل الأجراء فقط ولا تستخدم في البرنامج الرئيس وأن أي تغير على قيمة هذا المتغير سوف لا ينسحب على المتغيرات الأخرى التي تحمل نفس الأسم في الأجراءات الأخرى أو البرنامج الرئيس وهذه تسمى متغيرات محليه (local variables) .
ونوع أخر من المتغيرات من الممكن أن يستخدم داخل الأجراء والبرنامج الرئيس وأي تغير يطرأ على هذه المتغيرات ينسحب عليه في كل مكان وهي تسمى متغيرات عامه (global variables) .
المتغيرات المحليه يعلن عنها في حقل الأعلان عن المتغيرات داخل الأجراء بينما المتغيرات العامه يعلن عنها في حقل الأعلان عن المتغيرات في البرنامج الرئيس (أي أن أي متغير يعلن عنه في حقل الأعلان عن المتغيرات للبرنامج الرئيس سيكون فعالا ويرى (نستطيع قراءة وتغيير قيمته) في كل مكان يتواجد به في البرنامج , أما المتغيرات التي تعرف في حقل الأعلان عن المتغيرات داخلس الأجراء سوف لا ترى أطلاقا خارج هذا الأجراء) .

مثال :

```
Program CH5_Program1;
Var
  X:integer;

Procedure Change;
Begin
  X: =1;
End;      {change}

Begin     {main program}
  X: =0;
  Change;
  Write ('x=', x);
End.      {main program }
```

مخرجات البرنامج 1 ://

X = 1

شرح البرنامج (1) ://

هذا البرنامج يحتوي على متغير واحد هو (x) ومعرف في حقل الأعلان عن المتغيرات في البرنامج الرئيس لذ فهو يعتبر متغير عام (global) . البرنامج يحتوي على أجراء يقوم بتغيير قيمة المتغير (x) الى القيمه (1) .

يبدأ البرنامج الرئيس بأسناد القيمه (0) الى المتغير (x) , ثم يتم أستدعاء الأجراء (change) وذلك بكتابة أسم الأجراء عند ذلك ينتقل المؤشر الى الأجراء (change) وينفذ الأوامر التي فيه (كبرنامج مستقل) وبعد نهاية تنفيذ الأجراء يعود المؤشر الى عبارة أستدعاء الأجراء في البرنامج الرئيس ليستمر بالتنفيذ للعبارات التاليه لها والتي هي هنا عبارة الطباعه حيث ستطبع قيمة المتغير (x) .. قيمة المتغير تغيرت في الأجراء وتبعا لذلك تغيرت قيمتها في البرنامج الرئيس لأن المتغير معرف كمتغير عام .

```
Program CH5_Program2;
Var
  X:integer;

Procedure Change;
Var
  X: integer;
Begin
  X: =1;
End;      {change}

Begin     {main program}
  X: =0;
  Change;
  Write ('x=', x);
End.      {Main program}
```

مخرجات البرنامج ://

X = 0

شرح البرنامج (2) ://

لا يختلف هذا البرنامج عن البرنامج (1) كثيرا , عدا أن المتغير (x) عرف مرتان مره في حقل الأعلان عن المتغيرات في البرنامج الرئيس ليكون متغير عام , وثانية في حقل الأعلان عن المتغيرات في الأجراء (change) ليكون متغير محلي .

بناء على ما تقدم فأن المتغيرين يختلفان لأن أحدهما عام والثاني محلي وأن تشابهة التسميه, من ذلك فأن التغيير الذي يحصل على المتغير المحلي (x) داخل الأجراء سينحصر داخل الأجراء فقط ولا يؤثر على المتغيرات في البرنامج الرئيس سواء تشابهة التسميه أو أختلفت , ونفس الأمر ينسحب على المتغير العام .

ملاحظه ://

الوسائط (parameters) هي قيمه تمرر الى الأجراء أو الداله .
هناك أكثر من نوع من الوسائط وما يهمنا هنا نوعين :

1. **وسائط القيمه (value parameters) : تتصرف هذه الوسائط مثل المتغير المحلي ضمن الأجراء أو الداله (حيث أن المتغير المحلي يمكن تغيير قيمته ضمن الأجراء لكن القيمه الأصليه ضمن البرنامج الرئيس لاتتغير) , فعندما يستخدم أجراء وسائط قيمه فأن المترجم ينسخ هذه المتغيرات ويمرر نسخه منها الى الأجراء , أما المتغيرات الأصليه فتبقى على حالها لأن ما أرسل الى الأجراء هو نسخه وأي تغير في النسخه لا يؤثر في الأصل (كما لو نسخنا مستند وحورنا في النسخه المستنسخه فهل سيؤثر على النسخه الأصليه !؟) .**
وسائط القيمه تمرر من البرنامج الرئيس الى الأجراء فقط .
2. **الوسائط المرجعيه (Reference Parameters) : المتغيرات المرجعيه لا ترسل نسخه من المتغيرات بل ترسل المتغير ذاته , ولذا فأن أي تغير على هذا المتغير سيؤثر على المتغيرات الأصليه هذه المتغيرات تكون مسبوقه بالأمر (var) دائما .**
الوسائط المرجعيه تستخدم الطريقين (أي القيمه تنتقل من والى الأجراء) .

مثال :

أوجد قيمة (y) من المعادله التاليه :

$$Y = 10^{10} + 8^{8} + 6^{6} + 4^{4}$$

```
Program CH5_Program3 ;
Var
  Y, p1, p2, p3, p4: longint;
  i:integer;
Begin
  P1:=1;
  For i: =1 to 10 do
    P1:=p1*10;
  P2:=1;
  For i: =1 to 8 do
    P2:=p2*8;
  P3:=1;
  For i: =1 to 6 do
    P3:=p3*6;
  P4:=1;
  For i: =1 to 4 do
    P4:=p4*4;
  Y: =p1+p2+p3+p4;
  Writeln (p1,'  ', p2'  ', p3,'  ', p4);
  Writeln(y);
  Readln;
End.
```

مخرجات البرنامج ://

1410065408   16777216   46656   256
1426889536

ملاحظه ://

لو كان المتغير (y) معرف من نوع (integer) لكانت قيمة (y) تساوي (25792-) وذلك لأن النوع (integer) يحجز (2 Byte) وهي غير كافيه لتمثيل رقم أكبر من (65535) لذلك يتحول الى القيمه السالبه , وهو مشابه لعداد المسافه في السياره فعندما يصل الى القيمه العليا وهي (99999) فان الزياده بمقدار واحد تسبب بأن يكون العداد مساوي الى (00000) .

من الممكن أعادة كتابة البرنامج (3) بطريقة الأجراءات وكما يلي :

```
Program CH5_Program4;
Var
  P1, p2, p3, p4, y: longint;

Procedure power(x: integer; var p: longint);
Var
  I: integer;
Begin
  P: =1;
  For i: =1 to x do
    P: =p*x;
End;

Begin     {main program}
  Power (10, p1);
  Power (8, p2);
  Power (6, p3);
  Power (4, p4);
  Y: =p1+p2+p3+p4;
  Writeln(y);
  Readln;
End.
```

شرح البرنامج (4) ://

أولا : يبدأ البرنامج بتعريف عدد من المتغيرات والتي تعتبر عامه نظرا لأنها معرفه في حقل الأعلان عن المتغيرات للبرنامج الرئيس .

ثانيا : كتابة الأجراء والذي يسبق كتابة البرنامج الرئيس والأجراء يحتوي على قوس فيه متغيرات ونلاحظ أن هذه المتغيرات تأخذ شكلين : أحدهما مسبوق بالأمر (var) والفرق هنا ببساطه أن المتغير الأول يحمل قيمه تدخل الى الأجراء للعمل عليها داخل الأجراء , أما المتغير المسبوق بالأمر (var) فهو يعامل كمتغيرسيحمل قيمه تخرج من الأجراء الى البرنامج الرئيس والتي من الممكن أستخدامها في البرنامج الرئيس .

ثالثا : البرنامج الرئيس والذي سيتم أستدعاء الأجراء داخله لحساب القيم المطلوبه والأستدعاء يجب أن يتم من خلال كتابة أسم الأجراء المطلوب أستدعاءه ثم كتابة المتغيرات التي يراد أرسالها الى الأجراء هنا نرسل الرقم المطلوب حساب قيمته بعد رفعه الى الأس المطلوب وكذلك يكتب المتغير الذي ستعود النتيجه فيه (أن القيم الموجوده في أمر الأستدعاء ستنتقل الى الأجراء لتسند الى المتغير الذي يقابلها في الأجراء) .

رابعا : نتيجة رفع عدد الى أس معين وحسب الحالات أعلاه ستعود بالمتغير (p) والتي ستسند الى المتغيرات المقابله في أمر أستدعاء الأجراء ولكل حالة أستدعاء , هذه القيم تجمع لنحصل على النتيجه النهائيه .

ملاحظه ://

يجب أن يكون عدد الوسائط بين القوسين بعد أسم الأجراء في رأس الأجراء مساويا الى عدد الوسائط الموجوده بين القوسين في أمر أستدعاء الأجراء في البرنامج الرئيس .

ملاحظه ://

الوسائط بين القوسين في أمر الأستداء يجب أن تكون متناسبه مع الوسائط في رأس الأجراء , فالوسائط التي في رأس الأجراء أذا لم تكن مسبوقه بالأمر (var) فيجب أن يقابلها في أمر الأستدعاء أما قيمه ثابته أو متغير له قيمه محدده قبل ذلك . أما المتغير الذي في رأس الأجراء والمسبوق بالأمر (var) فيقابله في الأستدعاء متغير .
في الحاله الأولى ستنتقل القيمه من الأستدعاء الى الأجراء وكل قيمه تنتقل الى المتغير الذي يقابلها حسب ترتيبها (أي القيمه الأولى تسند الى المتغير الأول والقيمه الثانيه تسند الى المتغير الثاني وهكذا) .
أما الحاله الثانيه فتنتقل القيمه من داخل الأجراء الى المتغير في رأس الأجراء ثم منه تسند الى المتغير الذي يقابله في أمر الأستدعاء وحسب الترتيب أيضا .
كذلك يجب أن تتطابق الأنواع بين المتغيرات في أمر الأستدعاء وتلك في رأس الأجراء وكل واحد وما يقابله .

مثال :

```
Program CH5_Program5;
Var
  X:integer;

Procedure Change ( var y:integer);
Begin
  y:=1;
End;       {change}

Begin
  X: =0;
  Change(x);
  Write ('x=', x);
End.       {Main program}
```

مخرجات البرنامج 5 ://

X = 1

شرح البرنامج (5) ://

هذا البرنامج فيه متغير عام يدعى (x) ووسيط مرجعي يدعى (y) كتبنا هنا المتغير (y) فقط لتجنب الأشتباه والحقيقه هو أن البرنامج سيعمل بنفس الطريقه لو أستبدلنا (y) وأينما ترد (y) في البرنامج أعلاه بالمتغير (x) .
ولما كانت (y) في رأس الأجراء مسبوقه بالأمر (var) فهذا يعني أنها متغير وستنتقل القيمة المقابله من داخل الأجراء اليها (عند أستدعاء الأجراء وتنفيذه) , ثم منها الى أمر الأستدعاء (أي من y الى x) .

ملاحظه ://

عنما تكون الوسائط متغيرات (أي مسبوقه بالأمر var) في رأس الأجراء فأن المتغيرات المقابله في أمر الأستدعاء يجب أن تكون متغيرات وليس تعابير (expression) فمثلا الأستدعاء (change (2*x)) غير مقبول لأنه يستخدم التعبير (2*x) وكذلك الأستدعاء (change (2)) أيضا غير مقبول لأنه يستخدم ثابت (المفروض أن يكون متغير) .

```
Program CH5_Program6;
Var
  X:integer;

Procedure Change(y: integer);
Begin
  y:=1;
End;        {change}

Begin
  X: =0;
  Change (x);
  Write(x);
End.        {Main program}
```

مخرجات البرنامج :

X = 0

شرح البرنامج (6) ://

مرة أخرى , لدينا متغير عام (x) ووسيط (y) . في هذه الحاله الأعلان عن الوسيط غير مسبوق بالأمر (var) ولذلك فأن هذا الوسيط يدعى وسيط قيمه .
عند أستدعاء الأجراء فأن المتغير (x) يمرر الى الأجراء فيكون (y : = x) قبل أن يتم تنفيذ أي عباره من الأجراء . ولكن بعد الدخول في الأجراء فسوف لاتكون هناك أي علاقه بين المتغير (x) والمتغير (y) , عليه فأن التغيير الذي يطرأ على (y) سوف لايؤثرعلى المتغير (x) .

ملاحظه ://

عندما يكون الوسيط وسيط قيمه فمن الممكن أستخدام التعابير مثل (change (x*2) OR change (2)) لأن في الحاله الأولى ستكون هناك عملية أسناد ضمنيه (y : = 2 * x) وكذلك في الحالة الثانيه (y : = 2) , وقبل تنفيذ أي عباره من عبارات الأجراء .

مثال : نفرض أننا نرغب بأبدال قيم متغيرين بحيث تكون كل واحده في مكان الأخرى

```
Program CH5_Program7;
Var
  X,y:integer;

Procedure swap ( x,y:integer);
Var
  T:integer;
Begin
  T: =x;    x: =y;
  Y: =t;
  Writeln ('x=', x,'y=', y);
End;

Begin
  X: =3;
  Y: =42;
  Swap(x, y);
  Writeln ( 'x= ', x,' y= ', y);
End.
```

مخرجات البرنامج 7 ://

X = 42   y = 3
X = 3   y = 42

لم تتغير قيم (x, y) في البرنامج الرئيس بينما تغيرت في الأجراء
(تمرير وسيطة قيمه)

```
Program CH5_Program8;
Var
  X,y:integer;

Procedure swap ( var x,y:integer);
Var
  T:integer;
Begin
  T:=x;    x:=y;
  Y: =t;
  Writeln ( 'x= ', x,' y= ', y);
End;

Begin
  X: =3;
  Y: =42;
  Swap(x, y);
  Writeln ( 'x= ', x,' y= ', y);
End.
```

مخرجات البرنامج ://

X = 42   y = 3
X = 42   y = 3

تغير قيمتي (x, y) في البرنامج الرئيس والأجراء.

5.3 الدوال FUNCTIONS

أن الفرق الرئيس بين الأجراءات والدوال هو أن الدوال يجب أن تعيد قيمه عند تنفيذها بينما الأجراءات لاتعيد قيمه . تبدأ الدوال وتنتهي بنفس الطريقه التي تبدأ وتنتهي فيها الأجراءات . أما القيمه المعاده في الدوال فتعاد بأسم الداله , وهذا يعني أن أسم الداله سيكون متغير يحمل قيمه هذه القيمه هي القيمة الناتجه من تنفيذ الداله , وأزاء ذلك ولما كان أسم الداله هو متغير أذن يجب أن يكون له نوع ... عليه دائما سنلاحظ أسم الداله متبوع بنوعه .

الصيغه العامه للداله هي :

Function < function name > : type ;
OR
Function < function name > (Arguments) : type ;

مثال : في هذا المثال سنعيد كتابة البرنامج (3) ولكن مع أستخدام الدوال

```
Program CH5_Program9;
Var
  Y: longint;

Function power (m: integer): longint;
Var
  I:integer;
  P:longint;
Begin
  P: =1;
  For i: =1 to m do
    P: =p*m;
  Power: =p;
End;      {end function}

Begin     {main program}
  Y: = power (10) + power (8) + power (6) + power (4);
  Writeln(y);
  Readln;
End.
```

ملاحظه :

1. **كل الدوال والأجراءات تكتب بعد الأمر (var) في البرنامج الرئيس , وليس قبله .**
2. **أن أستدعاء الدوال والأجراءات في البرنامج الرئيس غير محدده بترتيب هذه الدوال أو الأجراءات .**
3. **من الممكن أن تتداخل الأجراءات أو الدوال (بحيث يمكن أن تستدعى أجراء أو داله من داخل داله أو أجراء أخر) .**
4. **عند حدوث التداخل فيجب مراعاة الترتيب حيث لا يجوز أستدعاء داله أو أجراء من داخل داله أو أجراء أخر يكون سابق له بالترتيب عند كتابة الأجراءات والدوال (يجب أن تكون المستدعاة سلبقه) .**

ملاحظه :

متى نستخدم الداله بدل الأجراء ؟
القاعده العامه هو أن نفكر أن الداله كهيكل تعيد قيمه واحده فقط . تستخدم الداله عندما يكون المطلوب قيمه مفرده في البرنامج الرئيس .

5.4 أمثله محلوله

- أكتب برنامج لأيجاد قيمة (y) , حيث أن

$$Y = x + x^2 / 2! + x^3 / 3! + \ldots\ldots\ldots + x^n / n!$$

```
Program CH5_Program10;
Var
  J, n: integer;
  Y: real;

Function power (x, j: integer): integer;
Var
  I, p: integer;
Begin
  P: =1;
  For i: =1 to j do
    P: =p*x;
  Power: =p;
End;

Function factorial (j:integer):integer;
Var
  F,i:integer;
Begin
  F: =1;
  For i: =j downto 1 do
    F: =f*I;
  Factorial: =f;
End;

Begin  {main program}
  Writeln ('Enter number of elements');
  Readln (n);
  Y: =0;
  For j: =1 to n do
    Y: =y + power(x, j) / factorial (j);
  Writeln(y: 6:4);
End.
```

- أكتب برنامج لأيجاد الرقم الأكبر في مصفوفه أحاديه حجمها (25) عنصر.

```
Program CH5_Program11;
Const
    N=25;
Var
  A: array [1..n] of integer;
  X: integer;

Procedure readarray;
Var
  I: integer;
Begin
  For i: =1 to n do
    Readln (a[i]);
End;

Procedure findmax (var max: integer);
Var
  I: integer;
Begin
  Max: =a [1];
  For i: =2 to n do
    If (a[i] > max) then
      Max: =a[i];
End;

Begin  {main program}
  Readarray;
  Findmax(x);
  Writeln(x);
End.
```

- أكتب برنامج لأيجاد العدد الأكبر بين ثلاثة أعداد .

```
Program CH5_Program12;
Var
  T,x,y,z:integer;

Function largest(x, y: integer): integer;
Begin
  If x > y then
    Largest: =x
  Else
    Largest: =y;
End;

Begin     {main program}
  Writeln ('Enter three numbers');
  Readln(x, y, z);
  T: =largest(x, y);
  Writeln (' largest number=', largest (t, z));
End.
```

- أكتب برنامج لأيجاد مربع أي عدد (حيث أن مربع العدد يساوي مجموع عدد من الأعداد الفرديه (عددهم بقدر العدد المراد أيجاد مربعه) أبتداءا من العدد واحد صعودا) (مثلا مربع العدد (4) هو (1+3+5+7) ويكون الناتج (16)) .

```
Program CH5_Program13;
Var
  Y: integer;
  Ch: char;

Procedure square (x: integer);
Var
  I, sum: integer;
Begin
  Sum: =0;   i: =1;
  Repeat
    Sum: = sum + I;
    Inc (I, 2);
    Dec(x);
  Until(x<=0);
  Writeln ('Square of number',y,'=',sum);
End;

Begin
  Writeln ('Enter number to find its square');
  Repeat
    Readln(y);
    Square(y);
    Writeln ('Do you want to enter another number(Y/N)');
    Readln (ch):
  Until (ch='N');
End.
```

الفصل السادس

السلاسل الحرفيه STRINGS

6.1 المقدمه

السلسله الحرفيه (String) هي عدد من الأشياء في خط , مثل سلسله دور , السلسله الحديديه (والتي هي مجموعه من الحلقات المتصله واحده بالأخرى) , وكذلك السلسله الحرفيه والتي هي مجموعه من الحروف .
لقد تعرفنا سابقا على ماهية متغير السلسله الحرفيه , ولكن مالم نتعلمه لغاية الأن هو ماهي العمليات والدوال التي ممكن أن تطبق على السلاسل الحرفيه وكيفية التعامل معها وتحويرها للحصول على سلاسل جديده من السلسله الأصليه .
في هذا الفصل سنغطي بعض الدوال المهمه والتي توفرها لنا لغة البرمجة باسكال .

6.2 ماهي السلاسل الحرفيه

لغرض الفهم الجيد للسلاسل الحرفيه , يجب أن نضع في أذهاننا بأن السلاسل الحرفيه هي عباره عن **مصفوفه** من الحروف . أن نوع البيانات للسلاسل الحرفيه هي (نوع بيانات مبني داخليا) مصفوفه من (256 حرف) :

```
Type
    String = array [ 0 .. 255 ] of char ;
```

لهذا السبب فأن المتغير الذي يعلن عنه في حقل الأعلان عن المتغيرات على أنه سلسله حرفيه فسيحجز له (256 بايت) كما سبق وبينا في الجدول (1.4) .
أذن أصبح واضح أن الذاكره ستحجز (256 بايت أو موقع متجاور) لمتغير السلاسل الحرفيه , لذلك فأن المعالج يجب أن يعرف موقع البدايه للسلسله الحرفيه وموقع النهايه أيضا (أي أين تبدأ السلسله الحرفيه وأين تنتهي) . ولما كانت السلسله الحرفيه مصفوفه حسب التعريف أعلاه فأن مواقع المصفوفه (1 .. 255) ستحتوي على الحروف التي تحتويها السلسله وهذا لايعني أن تكون السلسله بطول (255) حرف حيث من الممكن أن تكون أقل من ذلك لكن الطول الأعظم لها هو (255) . أما الموقع (0) في مصفوفة السلسله الحرفيه فأنه سيحتوي على طول السلسله الحرفيه (عدد الأحرف في السلسله الحرفيه) , وبذلك سيكون واضح للمعالج أين تنتهي السلسله الحرفيه من خلال قراءة الموقع (0) في مصفوفة السلسله الحرفيه . مثال

```
Program CH6_Program1;
Var
  St: string;
Begin
  St: = ' computer ';
  Writeln (st [2]);
  Writeln (st [5]);
End.
```

مخرجات البرنامج ://

```
c
p
```

ملاحظه ://

في المثال أعلاه تم حساب الفراغين الموجودين قبل وبعد (computer) على أنهما حروف , لذا دائما تحسب الفراغات على أنها حروف وفي أي موقع ترد في السلسله الحرفيه .

في هذا المثال تم التعامل مع السلسله الحرفيه على أنها مصفوفه دون الحاجه الى تعريفها كمصفوفه لأنها معرفه داخل الحاسب كمصفوفه كما بينا .

6.3 العمليات التي تجرى على السلاسل الحرفيه

هناك بعض الدوال المخزونه مع لغة البرمجه باسكال والتي تساعد على أجراء بعض العمليات على السلاسل الحرفيه دون الحاجه الى أعادة كتابتها ويكفي أستدعائها بأسمها لتقوم بالعمل المحدد لها . من هذه الدوال :

6.3.1 تحديد الموقع POS

نحتاج أحيانا الى تحديد موقع حرف أو مجموعة حروف داخل سلسله حرفيه , ولتحقيق ذلك نستخدم الداله (Pos) والتي ستقوم بالبحث عن (الحرف أو مجموعة الحروف والتي تعتبر مجموعه جزئيه من السلسله الحرفيه) داخل السلسله الحرفيه , فاذا لم يتم أيجاد السلسله الجزئيه ضمن السلسله الأصليه فأنها تعيد القيمه (0) أما أذا وجدته فأنها ستعيد الرقم الذي يحدد موقع أول حرف من السلسله الجزئيه .

الصيغه العامه لهذه الداله هي :

N : = Pos (st , stn) ;

حيث أن :

N : تمثل القيمه التي ستعيدها الداله.

st : تمثل السلسله التي نبحث عنها.

stn : تمثل السلسله التي نبحث فيها.

مثال :

```
Program CH6_Program2 ;
Var
  N, x: integer;
  Stn, st1, st2: string;
Begin
  Stn: = 'System Software';
  St1: = 'Soft';
  St2: = 'system';
  N: = Pos (st1, stn);
  X: = Pos (st2, stn);
  Writeln (n,' ****** ', x);
End.
```

مخرجات البرنامج: //

```
8 ****** 0
```

لاحظ أن قيمة المتغير (x) تساوي (0) وذلك لأن السلسله الحرفيه التي نبحث عنها هي (system) تبدأ بحرف صغير بينما الكلمه المقابله في (stn) (System) تبدأ بحرف كبير وبذلك أصبح أختلاف بين الكلمتين لذلك عملية البحث سوف لاتجد السلسله الحلرفيه المطلوبه , أذن تعيد القيمه (0) .

6.3.2 الأستنساخ COPY

هذه الداله ستقوم بأستنساخ حرف أو مجموعة حروف من السلسله الحرفيه المحدده , أعتبارا من الموقع المحدد من قبل المستخدم وحسب عدد الحروف التي حددها المستخدم , وستعيد هذه الداله سلسله حرفيه جزئيه وهي التي حددها المستخدم .

الصيغه العامه لهذه الداله هي :

St : = Copy (stn , Position , count) ;

حيث أن :

St : تمثل السلسله الحرفيه التي ستعيدها الداله.

Stn : تمثل السلسله الحرفيه المطلوب الأستنساخ منها.

Position : رقم يمثل موقع بداية السلسله المطلوب أستنساخها.

Count : رقم يمثل عدد الأحرف المطلوب أستنساخها.

مثال :

```
Program CH6_Program3;
Var
  Stn, st1, st2: string;
Begin
  Stn:= 'computer science';
  St1:= copy (stn, 2, 4);
  St2:= copy (stn, 12, 20);
  Writeln ( st1,' ****** ', st2);
End.
```

مخرجات البرنامج ://

```
ompu ******  ience
```

ملاحظه ://

لاحظ أن المتغير (st2) كان من المفروض أن يستنسخ فيه (20) حرف أبتداءا من الحرف رقم (12) ونظرا الى أن السلسله تنتهي قبل تحقق هذا الشرط فأنه سيكتفي بالمتبقي من الحروف .

6.3.3 الحذف DELETE

دالة الحذف تستخدم لحذف حرف أو عدد من الحروف من سلسله حرفيه معينه , وبعد الحذف يعيد السلسله الحرفيه الجديده وبنفس متغير السلسله الحرفيه الأصليه , أن الموقع الذي يبدأ منه الحذف وعدد الحروف المطلوب حذفها تحدد من قبل المستخدم.

الصيغه العامه لهذه الداله هي :

Delete (stn, Position, Count) ;

حيث أن :

Stn : السلسله الحرفيه المطلوب الحذف منها.

Position : رقم يمثل الموقع الذي يبدأ الحذف منه.

Count : رقم يمثل عدد الحروف المطلوب حذفهم.

مثال :

```
Program CH6_Program4;
Var
  Stn: string;
Begin
  Stn: ='System Software';
  Delete (stn, 8, 3);
  Writeln (stn);
  Delete (stn, 7, 15);
  Writeln (stn);
End.
```

مخرجات البرنامج ://

System tware
System

ملاحظه ://

في أمر الحذف الثاني كان المطلوب حذف (15) حرف ابتداءا من الموقع (7) ونظرا لعدم وجود (15) حرف بعد الحرف السابع لذا سيتم حذف المتبقي من الحروف فقط .

6.3.4 الحشر INSERT

تعمل هذه الداله على حشر سلسله حرفيه بأي طول داخل سلسله حرفيه أخرى بدءا من الموقع المحدد من المستخدم (موقع الحرف في السلسله المصدر الذي سيتم الحشر بعده) . سوف لا يتم حذف أي من الحروف من الأمام عدا أذا كان طول السلسله الناتجه أكبر من (255) فعند ذلك ستحذف الحروف الزائده عن هذا العدد من الأمام .

الصيغة العامه لهذه الداله هي :

```
Insert ( st , stn , Position ) ;
```

حيث أن :

stn : تمثل السلسله الحرفيه الأصليه التي سيحشر فيها.

st : تمثل السلسله الحرفيه المطلوب حشرها.

Position : تمثل الموقع الذي ستحشر فيه السلسله الحرفيه.

مثال :

```
Program CH6_Program5;
Var
  S: String;

Begin
  S: = 'Hey! How are you?'
  Write (Insert (' Ali', s, 4));
  Writeln(s);
End.
```

مخرجات البرنامج ://

'Hey Ali! How are you?'

6.3.5 دمج سلسلتان حرفيتان CONCAT

تستخدم هذه الدالة لدمج سلسلتين حرفيتين أو أكثربسلسله حرفيه واحده.
الصيغة العامه لهذه الداله هي :

Stn : = Concat (st1, st2, st3, …) ;

حيث أن :

Stn : تمثل السلسله الناتجه من الدمج.
st1, st2, st3 : تمثل السلاسل التي سيتم دمجها.

مثال :

```
Program CH6_Program6;
Var
  Stn, st1, st2: string;
Begin
  St1:='info';
  St2:='rmation';
  Stn: =concat (st1, st2);
  Writeln (stn);
End.
```

مخرجات البرنامج ://

information

ملاحظه ://

ممكن الأستعاضه عن الداله (concat) بعلامة الجمع (+) .

يمكن أعادة كتابة البرنامج أعلاه ليكون :

```
Program CH6_Program7;
Var
  Stn, st1, st2: string;
Begin
  St1:='Bag';
  St2:='hdad';
  Stn: = st1 + st2;
  Writeln (stn);
End.
```

مخرجات البرنامج ://

Baghdad

6.3.6 حساب طول السلسله الحرفيه LENGTH

تستخدم هذه الداله لحساب طول السلسله الحرفيه.
الصيغه العامه لهذه الداله هي :

N : = Length (st) ;

حيث أن :

N : يمثل طول السلسله الحرفيه.
St : يمثل السلسله الحرفيه المطلوب أيجاد طولها.

مثال :

```
Program CH6_Program8;
Var
  St: string;
  N: integer;
Begin
  St: ='Data Structure';
  N: =Length (st);
  Write (n);
End.
```

مخرجات البرنامج ://

14

6.3.7 تحويل الحروف الصغيره الى حروف كبيره UPCASE

تعمل هذه الداله على تحويل الحروف الصغيره الى حروف كبيره , تحول حرف واحد كل مره . أما أذا كانت الحروف هي أصلا كبيره أو كانت خارج نطاق مفهوم الحروف الصغيره فستترك على حالها .

الصيغه العامه لهذه الداله هي :

C : = UpCase (c) ;

حيث أن :

C : الحرف الناتج وهو من الحروف الكبيره.

c : الحرف المطلوب تحويله الى حرف كبير وهو من الحروف الصغيره.

مثال :

```
Program CH6_Program9;
Var
  S: String;
  i: Integer;

Begin
  S: = 'Hey! How are you?';
  For i: = 1 to length(S) do
    S[i]:= UpCase(S[i]);
  Write(S);
End.
```

مخرجات البرنامج ://

HEY! HOW ARE YOU?

ملاحظه ://

لتحويل الأحرف الكبيره الى أحرف صغيره نتبع العلاقه التاليه :

S[i] : = chr (ord (s[i] + 32) ;

6.3.8 تحويل القيم الرقميه الى سلسله حرفيه STR

تعمل هذه الداله على تحويل الأعداد الصحيحه فقط الى سلاسل حرفيه . أما أذا أستخدمنا أرقام كسريه فستعيد لنا الداله القيمه صفر.

الصيغه العامه لهذه الداله هي :

STR (NUM , ST) ;

حيث أن ك

NUM : يمثل العدد المطلوب تحويله الى سلسله حرفيه.

ST : يمثل السلسله الحرفيه الناتجه.

مثال :

```
Program CH6_Program10;
Var
  X: integer;
  St: string;
Begin
  X: =2345;
  STR(x, st);
  Writeln(s);
End.
```

مخرجات البرنامج ://

'2345'

6.3.9 تحويل السلاسل الحرفيه الى أرقام VAL

تعمل هذه الداله على أيجاد القيمه العدديه للسلسله الحرفيه التي تحتوي على أرقام فقط على شكل حرفي .

الصيغه العامه لهذه الداله هي :

VAL (stn, number , Error) ;

حيث أن :

Stn : تمثل السلسله الحرفيه.

Number : يمثل المتغير الذي ستوضع فيه القيمه العدديه المتحوله.

Error : متغير يعيد عدد صحيح) عندما يتم التحويل دون أخطاء فستعيد القيمه (0) للدلاله على عدم وجود خطأ , أما أذا حدث خطأ فستعيد عدد صحيح يبين موقع الخطأ).

مثال :

```
Program CH6_Program11;
Var
  St: string;
  N, e: integer;
Begin
  St: ='1234';
  Val (st, n, e);
  Writeln (n,' ****** ', e);
  St: ='2345.12';
  Val (st, n, e);
  Writeln (n,' ^^^^^^ ', e);
End.
```

مخرجات البرنامج ://

```
1234 ****** 0
   0  ^^^^^^ 5
```

ملاحظه ://

الداله (val) لا تتعامل مع الكسور.

ملاحظه ://

يفضل أن يتم تحديد طول السلسله الحرفيه عند الأعلان عنها وذلك لترشيد مسا حة الذاكره المستخدمه , وكمثال ما يلي

name : string [30] ;

هنا حددنا المواقع التي تحجز في الذاكره للمتغير (name) بحجم (30) حرف بدلا من تركها دون تحديد وبالتالي تحدد من قبل المترجم بحجم (256) موقع (يراعى الرقم الذي نحدده بما يتناسب والحد الاعلى الذي نحتاج اليه لتمثيل السلسله الحرفيه) أما أذا كان من الصعب تخمين الحد الأعلى فيترك دون تحديد

6.4 أمثله محلوله

- أكتب برنامج لقراءه سلسله حرفيه ثم أطبعها بشكل معكوس (مثلا أذا كانت السلسله الحرفيه ABCD تطبع DCBA) .

```
Program CH6_Program12;
Var
  S, st: string;
  I: integer;
Begin
  Writeln ('Enter string');
  Readln(s);
  St: ='';
  For i: =length(s) downto 1 do
    St: = st + s[i];
  Writeln (st);
End.

  OR
     For i: =1 to length(s) do  {instead of for loop above}
       St: =s[i] + st;
```

- أكتب برنامج لقراءة سلسله حرفيه وجد عدد المرات التي يتكرر فيها الحرف (e) .

```
Program CH6_Program13;
Var
  S:string;  k,n,i:integer;
Begin
  Writeln ('Enter string');
  Readln(s);
  K: =0;
  N: =length(s);
  For i: = 1 to n do
    If(s[i] ='e') then
      K: =k+1;
  Writeln ('number of letter e =', k);
End.
```

- أكتب برنامج لفصل سلسله حرفيه الى ثلاث مجاميع واحده للحروف الكبيره والثانيه للأرقام والثالثه للحروف المتبقيه من السلسله الحرفيه .

```
Program CH6_Program14;
Var
  S, s1, s2, s3: string;
  I: integer;
Begin
  Writeln ('Enter string');
  Readln(s);
  S1:='';   s2:='';    s3:='';
  For i: = 1 to length(s) do
    Case s[i] of
      'A'..'Z': s1:=s1+s[i];
      '0'..'9': s2:=s2+s[i]
      Else  s3:=s3+s[i];
    End;
  Writeln ('Capital letters are:', s1);
  Writeln ('Digits are:' , s2);
  Writeln ('Rest of string are:', s3);
End.
```

- أكتب برنامج لقراءة سلسله حرفيه ثم أحذف كافة الفراغات الموجوده فيها .

```
Program CH6_Program15;
Var
  S: string;   x: integer;
Begin
  Writeln ('Enter string');
  Readln(s);
  X: =pos ('', x);
  While(x<>0) do
    Begin
      Delete(s, x, 1);
      X: =pos ('', s);
    End;
  Writeln(s);
End.
```

- أكتب برنامج لقراءة سلسله حرفيه ثم أحذف كل الكلمات (are) في السلسله الحرفيه .

```
Program CH6_Program16;
Var
  S:string;   x:integer;
Begin
  Writeln ('Enter string');   Readln(s);
  X: =pos ('are', s);
  While(x<>0) do
    Begin
      Delete(s, x, 3);
      X: =pos ('are', s);
    End;
  Writeln(s);
End.
```

- أكتب برنامج لقراءة سلسله حرفيه ثم حول جميع الأحرف الكبيره الى حروف صغيره وجميع الأحرف الصغيره الى حروف كبيره .

```
Program CH6_Program17;
Var
  S: string;   i: integer;
Begin
  Writeln ('Enter string');
  Readln(s);
  For i: = 1 to length(s) do
    Case s[i] of
      'a'..'z': s[i]:=upcase(s[i]);
      'A'..'Z': s[i]:=Chr (ord(s[i]) +32);
    End;
  Writeln(s);
End.
```

- أكتب برنامج لقراءة سلسله حرفيه ثم غير كل فراغ في السلسله الحرفيه الى (@@@).

```
Program CH6_Program 18;
Var
  S:string;
  N:integer;
Begin
  Writeln ('Enter string');
  Readln(s);
  N:=pos('',s);
  While(n<>0)do
    begin
      Delete(s, n, 1);
      Insert ('@@@', s, n);
      N: =pos ('', s);
  Writeln(s);
End.
```

- أكتب برنامج لقراءة سلسله حرفيه تتكون من مجموعه من الكلمات يفصل بين واحده وأخرى فراغ , المطلوب طباعة كل كلمه على سطر منفرد .

```
Program CH_6Program19;
Var
  Stn, st: string; x: integer;
Begin
  Writeln ('Enter string');
  Readln (stn);
  X: =pos ('', stn);
  While(x<>0) do
    Begin
      St: =copy (stn, 1, x-1);
      Writeln (st);
      Delete (stn, 1, x);
      X: =pos ('', stn);
   End;
   Writeln (stn);
End.
```

الفصل السابع

متغيرات الأنواع VARIABLES TYPE

7.1 المقدمه

سبق وأن أطلعنا على الأنواع القياسيه في لغة البرمجه باسكال مثل (Integer , Real , Boolean , char …etc) , مواصفات هذه الأنواع تحدد بشكل كامل من قبل المترجم عند تنفيذ برامج لغة البرمجه باسكال .
في هذا الفصل سنقدم أنواع أكثر, تحدد مواصفاتها من قبل المبرمج , سيتم تعريف أنواع مناسبه لكل برنامج مما يساعد على توضيح المشكله ويسهل قراءة وكتابة البرنامج .

7.2 الأنواع TYPES

تستخدم كلمة نوع (Type) للتصريح عن أنواع جديده من الممكن أستخدامها في البرنامج .
الصيغه العامه لأستخدام النوع هي :

```
Type
     Typename  = new type ;
```

لتوضيح هذه الصيغه نفرض أن البرنامج الذي نكتبه يحتاج الى مصفوفه (مجموعه من القيم متكونه من 20 قيمه) , وهذا النمط من الصفيفه سيستخدم في البرنامج مرارا وتكرارا , فيمكن الأعلان أو التصريح عن نمط جديد كما يلي :

```
Type
    MyArray  = array [ 1 .. 20 ] of byte ;
```

الأن يمكن أن نعلن عن متغيرات من نوع (myarray) بدلا من كتابة تعريف المصفوفه أعلاه في كل مره نحتاج الى مثل هذه المصفوفه .

```
Var
    X : myarray ;
```

الأعلان عن الأنواع الجديده سيكون في قسم الأعلان عن الأنواع والذي يسمى (Type) , وموقع هذا القسم في البرنامج يكون بين قسم الأعلان عن الثوابت وقسم الأعلان عن المتغيرات .

> **ملاحظه ://**
>
> **قسم الأعلان عن الأنواع الذي يظهر في جسم الأجراءات هي محليه لتلك الأجراءات .**
> **أما تلك التي تظهر في البرنامج الرئيس فهي عامه .**

بشكل عام هناك ثلاث أصناف للأنواع :

7.2.1 الأنواع العدديه SCALARS TYPE

الأنواع العدديه هي ببساطه قائمه من القيم , والتي من الممكن أن تفرض للمتغير قيمه من ذلك النوع , فمثلا أن النوع القياسي (integer) يمثل قائمه بالأعداد الصحيحه وعندما نعرف متغير من هذا النوع فأننا من الممكن أن نسند للمتغير أي قيمه من قيم الأعداد الصحيحه (فعندما نقول أن المتغير من النوع الفلاني فهذا يعني أن هذا المتغير سيأخذ قيمه محدده بذلك النوع) . فمثلا

```
Type
    Units = ( inches , feet , miles ) ;
```

لاحظ أن هذا النوع حدد بثلاث وحدات لذلك فأن المتغير الذي سيعرف من هذا النوع سوف لن يكون بمقدوره أن يحمل قيمه غير تلك المحدده بالتعريف (ممكن أن يأخذ أي قيمه من القيم الثلاث المحدده فقط) .

الأن بعد الأعلان عن نوع جديد غير موجود أصلا ضمن الأنواع القياسيه المحدده بلغة البرمجه باسكال , فأننا من الممكن أن نعلن عن متغير في حقل الأعلان عن المتغيرات من النوع (units) , وكما نعمل مع المتغيرات القياسيه , وسيكون هذا النوع فعالا ضمن هذا البرنامج فقط وليس البرامج الأخرى , فأذا ما كانت هناك حاجه لأستخدام هذا النوع في برنامج أخر فيجب أن نعلن عنه ضمن البرنامج الجديد .

```
Var
    X : units ;
```

ملاحظه ://

التعريفان (الأعلان عن النوع والأعلان عن المتغير من ذلك النوع) من الممكن أن يدمجان معا بتعريف واحد كما يلي :

```
Var
    Scale : ( inches , feet , miles ) ;
```

ولكن في معظم الحالات يفضل أن يفصل بين تعريف الأنواع وتعريف المتغيرات .

أمثله عن الأنواع العدديه :

```
Type
     Day = ( Monday, Tuesday, Wednesday, Thursday, Friday,
                  Saturday, Sunday ) ;
     Operator = ( plus, minus, multiply, divide ) ;
    Trigfunction = ( sine, cosine, tangent, secant, cosecant ) ;
```

هنا سنعرض متغيرات من تلك الأنواع

```
Var
```

```
Holiday, workday   :  day ;
Addingop, multop   :  operator ;
```

ملاحظه ://

لسوء الحظ فأن لغة البرمجه باسكال لا توفر أمكانية قراءة وطباعة القيم العدديه مباشره (المتغيرات من هذا النوع) . مثلا

```
Holiday := Friday;
Write ( holiday )
```

سوف لا يطبع (Friday) , وربما لانحصل على أي مخرجات لأن المترجم سيصدر رسالة خطأ .

ملاحظه ://

لاحظ أن قيم الأنواع عادة تكون محدده بين قوسين , ولا يجوز أن تكون هناك قيمه تعود الى نوعين في ذات البرنامج , يجب أن تحدد لنوع واحد , فمثلا تعريف الأنواع التاليه غير مقبول :

```
Type
     Fruit = ( apple, orange, lemon, pineapple, tomato );
    Vegetable = ( potato, carrot, tomato, onion ) ;
```

والسبب لأن العنصر (tomato) يظهر في التعريفين .

أن القيم المدونه في الأعلان عن النوع العددي ثابته لذلك النوع , لذلك من الممكن أن نكتب

```
Holiday : = Friday ;                    scale : = miles ;
```

ملاحظه ://

لايجوز أسناد قيمه من نوع معين (أستخدام المساواة) الى متغير من نوع أخر . مثلا :

```
Holiday := miles ;
```

- أن العمليات التي من الممكن أستخدامها مع المتغيرات العدديه هي العمليات العلائقيه (relational operators) وطبعا ستكون نتيجة هذا التعبير هي من النوع (Boolean) أي صح أو خطأ . فمثلا

```
Monday  <  Friday
Multiply  >  divide
```

أن النوع (Boolean) هو بحد ذاته نوع عددي , فهو معرف ضمنيا كنوع قياسي كما يلي :

```
Type
     Boolean = ( false , true );
```

وطبعا أذا قلنا

```
False < true
```

وهي عباره صحيحه .

- الدوال (succ , pred) هي معرفه للأنواع العدديه (أي يمكن أستخدامها مع الأنواع العدديه) , والقيمه المعاده هي من نفس نوع المعاملات المستخدمة , مثلا

```
Succ ( Monday ) = Tuesday
```

مع ملاحظة أن أول عنصر في القائمه ليس له (pred) , وأخر عنصر في القائمه ليس له (succ) .

- الداله (ord) لها معاملات عدديه وتعيد قيمة عدد صحيح , والذي هو العدد الترتيبي للقيمه العدديه في تعريف القائمه حيث أن أول قيمه في القائمه لها العدد الترتيبي صفروالذي يليه له عدد ترتيبي قيمته واحد وهكذا .. مثلا.. ومن الأنواع أعلاه :

```
Ord ( plus ) = 0
Ord ( tuesdy ) = 1
Ord ( cosecant ) = 4
```

- أيضا من الممكن أستخدام أمر التكرار (for) بالهيكليه :

```
For scale : = inches to miles do
          Do something ;
```

هنا لا يمكن أستخدام (while) مثلا :

```
Scale : = inches ;
While ( scale <= miles ) do
Begin
     Do something ;
   Scale: = succ ( scale ) ;
```

```
End;
```

سيفشل هذا البرنامج عند حساب (succ (miles)) .

7.2.2 المديات الجزئيه SUBRANGE

هذا النوع يعرف بأسناد ثابتين للمتغير في حقل الأعلان عن النوع , وهذان الثابتان يمثلان الحدود الدنيا والحدود العليا للمدى الجزئي , ويجب أن يختلفان ويكونان من نفس النوع .مثال :

```
Type
      Index = 1 .. 35 ;
```

هنا الحد الأدنى هو (1) والحد الأعلى هو (35) والأثنان من نوع الأعداد الصحيحه . والمدى يكون بين (35 – 1) , عليه فأن المتغير الذي يعرف من نوع (index) سيأخذ أي قيمه من القيم المحدده بالمدى (35 – 1) وليس قيمه خارج هذا المدى .

> **ملاحظه ://:**
>
> **لا تستخدم الأعداد الحقيقيه مع هذا النوع (أي لا يمكن أن نستخدم الأعداد الحقيقيه في المديات الجزئيه) .**

- التعريفان (الأعلان عن النوع والأعلان عن المتغير من ذلك النوع) من الممكن ان يدمجان معا بتعريف واحد .. مثال

```
Type
    Index = 1 .. 35 ;
    Letter = 'a' .. 'z' ;
    Digit  = '0' .. '9' ;
 Var
     Count : index ;
    Firstchar , lastchar : letter ;
```

هذان التعريفان من الممكن دمجهما بتعريف واحد كما يلي :

```
Var
   Counter : 1 .. 35 ;
   Firstchar , lastchar : 'a' .. 'z' ;
```

لكن في معظم الحالات يفضل المحافظه على تعريفين منفصلين .

ملاحظه ://:

- **المعاملات (operators) التي من الممكن أن تستخدم مع متغير لنوع معين , ممكن أيضا أن تستخدم مع المدى الجزئي لهذا النوع . مثال .. أذا كان :**

```
Var
   Index : 1 .. 10 ;
   Number : 1 .. 100 ;
   Result : integer ;
```

فأن هذه العباره هي عباره صحيحه :

```
   Result + number div index
```

- **متغيرات المدى الجزئي ممكن أيضا أستخدامها على كلا جانبي المساواة . مثال**

```
  index  : =  number ;
  result : = index ;
```

7.2.3 المجموعات SETS

المجموعه هي تجميع لأشياء من نفس النوع , فأذا كانت (S) مجموعه من الأشياء من نوع (t) فأن أي مكون أو أي شيء من النوع (t) أما أن يكون عضو (عنصر) في المجموعه (S) أو لا يكون عنصر في المجموعه (S) .
من الممكن أن نعرف نوع المجموعه وفقا لأي نوع عددي (أن نوع القيم في المجموعه هي مجاميع من النوع العددي) . مثال

```
Type
 Ingredients = ( apple , orange , bananas , strawberries , nuts ,
                        icecream , cream , pastry , sugar , ice ) ;
```

تعريف المجموعه يكون :

```
Type
   Desert = set of ingredients;
```

أما المتغيرات من نوع (desert) فيعلن عنها في حقل الأعلان عن المتغيرات :

```
Var
   Feast , x : desert ;
```

القيم الثابتـه أو المتغيره من النـوع (desert) هي مجموعـه جزئيـه من النـوع (ingredients) والتي تعتبر النوع الأساس للنوع (desert) .
تمثل المجموعـه بقائمـه من العناصـر المعرفـه ضـمن المجموعـه , تفصل فـارزه بـين عنصـر وأخر , وجميع هذه العناصر محدده بقوسين مربعين . مثال

[icecream , cream]
[apple , bananas , icecream]
[orange]

ملاحظه ://

أذا كانت عناصر مجموعه معينه متعاقبه فيمكن تحديد العنصر الأول والأخير فقط كمدى جزئي . مثال
[apple , orange , bananas , strawberries]
من الممكن أن تكتب :
[apple .. strawberries]

صفات المجموعات

- من الممكن أن لا تحتوي المجموعـه علـى أي عنصـر عنـدها تسـمى مجموعـه خاليـه , ويرمـز للمجموعه الخاليه بالرمز [ ] .
- أتحـاد (union) أثنـان مـن المجـاميع ينـتج عنهمـا مجموعـه واحـده تحتـوي عناصـر المجموعتين . عامل الأتحاد هو (+) . مثال

[apple] + [bananas , sugar] = [apple , bananas , sugar]

- تقاطع (intersection) أثنان من المجاميع ينتج عنه مجموعه واحده تحتوي فقط العناصـر المشتركه بـين المجموعتين (أي العناصـر الموجـوده فـي المجموعـه الأولـى وبـذات الوقـت موجوده في المجموعه الثانيه) . عامل التقاطع هو (*) . مثال

[Orange , icecream , cream] * [icecream , nuts] = [icecream]

- الفرق بين أثنان من المجاميع ينتج عنه مجموعه واحده فقط تحتـوي علـى كل العناصـر التـي في المجموعه الأولى ولكنها ليست عناصـر فـي المجموعـه الثانيـه . عامل الفرق هـو (-) . مثال

[apple , strawberries , bananas] – [strawberries , cream] = [apple , bananas]

- الرمز (IN) هي من الكلمات المحجوزه تستخدم هنا لأختبارأنتماء عنصر لمجموعه , وطبعا نتيجة ذلك هو عباره منطقيه (صح أو خطأ) , **مثال** العباره التاليه عباره صحيحه

Apple in [apple , orange , icecream]

بينما العباره التاليه هي عباره خاطئه

Apple in [orange , bananas , icecream , cream]

- تستخدم العوامل العلائقيه للمقارنه بين المجاميع , والمبينه بالجدول (7.1) .. **أمثله** على ذلك :

[Orange , cream] = [cream , orange]
[icecream] ≠ [ice , cream]
[strawberries] <= [strawberries , cream]
[apple .. ice] > = [icecream .. cream]

- عبارة المساواة ربما من الممكن أستخدامها مع متغيرات المجاميع وتعابير المجموعه .. **مثال**

X : = [apple , cream , sugar] ;
Feast : = x + [icecream] ;

جدول (7.1) : مقارنة العمليات العلائقيه المستخدمه في لغة البرمجه باسكال مع ما يقابلها في الرياضيات

الرمز في لغة البرمجه باسكال	الرمز المستخدم في الرياضيات	المعنى
[…]	{ … }	المجموعه (set)
+	U	الأتحاد (union)
*	∩	التقاطع (intersection)
≤	≤	تحتوى (contains)
≥	≥	محتواة بالمجموعه (contained by) (مجموعه جزئيه من)
IN	ϵ	تعود الى (inclusion)
[ ]	Ø	مجموعه خاليه (empty set)

ملاحظه ://

عند أستخدام العلامه ($\leq$ أو $\geq$) فأن فتحة هذه العلامه دائما تشير الى المجموعه الأكبر .

7.3 أمثله محلوله

- أكتب برنامج لتكوين مجموعتين ذات أعداد (0 .. 255) ثم أوجد مايلي :
  1. أطبع المجموعتين
  2. جد أتحاد المجموعتين وأطبع النتيجه
  3. جد تقاطع المجموعتين وأطبع النتيجه
  4. جد متممه المجموعه الأولى والثانيه وأطبع النتيجه
  5. جد العناصر الموجوده في المجموعه الأولى وغير موجوده في المجموعه الثانيه وأطبع النتيجه.

```
Program CH7_Program1;
Type
   Number=set of 0..255;
Var
   U, t1, t2, t3, t4, t5, t6, t7: number;

Procedure readset (var s1: number);
Var
  I, x: integer;
Begin
  S1:= [];
  For i: =1 to 10 do
    Begin
      Writeln ('Enter number 0..255');
      Readln(x);
      S1:=s1+[x];
    End;
End;

Procedure writeset (s1: number);
Var
  I: integer;
Begin
  Write ('{');
  For i: = 0 to 255 do
    If (I) IN (s1) then    write (i: 4);
  Write ('}');
End;

Begin
  Readset (t1);
  Readset (t2);
  Writeset (t1);
  Writeset (t2);
  T3:=t1+t2;    {union}
  Writeset (t3);
  T4:=t1*t2;    {intersection}
  Writeset (t4);
  U:=[0..255];
  T5:=u-t1;
  Writeset(t5);
  T6:=u-t2;
  Writeset (t6);
  T7:=t1-t2;
  Writeset (t7);
End.
```